TÉCNICO LISBOA

academia militar

ROVIM T2D To do (1) To do (2)

Um veículo autonomo de vigilância de instalações militares

Gonçalo Filipe Ribeiro André

Dissertação para a obtenção do grau de mestre em

Engenharia Eletrotécnica e de Computadores

Orientadores: Prof.To do (3) António Joaquim Serralheiro
Prof. Duarte de Mesquita e Sousa

Júri

Presidente: Prof. Lorem
Orientador: Prof. António Joaquim Serralheiro
Co-Orientador: Prof. Duarte de Mesquita e Sousa
Vogais: Dr. Lorem Ipsum
Prof. Lorem Ipsum

Abril 2016

Agradecimentos

Gostaria de agradecer, à minha família pela oportunidade que me deu de prosseguir os meus estudos, aos meus amigos pelo apoio que me deram e pelos bons momentos que passámos durante a vida académica, à minha namorada por me ter aturado e continuar a fazê-lo, e ao meu orientador por me ensinar a ser um engenheiro melhor.

Abstract

The Objective of this Work ... (English)

Keywords

Keywords (English)

Resumo

O objectivo deste trabalho ... (Português)

Palavras Chave

Palavras-Chave (Português)

Conteúdo

Lista de Figuras

Lista de Tabelas

Siglas e Acrónimos

To do (4)

ROVIM Robô de Vigilância de Instalações Militares

T2D Tração, Travagem e Direção

MDF *Medium-Density Fibreboard*, fibra de madeira de média densidade

DC *Direct Current*, corrente contínua

AC *Alternating Current*, corrente alternada

NiMH *Nickel Metal Hydride*, níquel-hidreto metal

VRLA *Valve-Regulated Lead-Acid*, bateria de ácido chumbo selada

Li-Ion *Lithium-ion*, iões de lítio

PID Proporcional Integral Derivativo

1

Introdução

Resumo da dissertação para quem não a leu.

- introduzir o projeto: retirar do enquadramento da tese;
- porquê é que se está a fazer este projeto.
- Descrever a plataforma ROVIM
- descrever o T2D
- descrever com os outros módulos interagem com o T2D. Qual o papel do T2D na plataforma completa.
- Apresentar claramente o objetivo deste projeto;
- descrever sucintamente o que foi feito: abordar o problema, planear e construir, e observar os resultados;
- resumir os capitulos.
- sem imagens ou tabelas.

2

Enquadramento

2.1 Análise do caderno de encargos

O objectivo deste projeto é a construção de um protótipo funcional de um veículo autónomo destinado à vigilância de instalações militares. A sua estrutura básica é definida no enquadramento do módulo Tração, Travagem e Direção (T2D) e consiste em motores elétricos de tração, travagem e direção e seus controladores, baterias e sistema de monitorização, embarcados no *chassis* de uma moto-quatro.

Os requisitos funcionais da plataforma para este módulo são apresentados na tabela 2.1. Esta não pretende ser uma lista exaustiva, mas apenas balizar o desenho da plataforma. Estes foram definidos a partir do âmbito do projeto e de uma análise das necessidades e potenciais capacidades de uma plataforma deste género.

Os requisitos funcionais são demasiado vagos para orientarem eficazmente o desenvolvimento do projeto. Mas não faz sentido criar demasiadas condicionantes numa prova de conceito e primeira iteração de um protótipo executado por alunos sem experiência prévia neste tipo de trabalho. Por isso foram definidos princípios de desenho, que pretendem emular requisitos subjacentes à ideia da plataforma. Este princípios permitem orientar o desenvolvimento do protótipo e eliminar a rigidez de uma longa lista de requisitos mensuráveis, muitos dos quais não são conhecidos à partida.

É expectável nesta iteração do protótipo que algumas das soluções inicialmente projetadas se mostrem inviáveis e tenham que ser corrigidas. É também razoável esperar que o protótipo apresentado seja alvo de modificações em iterações futuras. Assim, é essencial que todas as soluções projetadas acomodem facilmente alterações futuras. Por isso, estas devem ser simples, flexíveis e sobredimensionadas. As optimizações devem ser relegadas para iterações futuras.

O protótipo a desenhar deverá ser tão seguro quanto possível, quer ao nível da prevenção de curto-

circuitos e choques elétricos, quer ao nível da prevenção de embates. As suas dimensões tornam-no capaz de infligir sérias lesões em caso de acidente.

Parâmetros como a fiabilidade, a autonomia, o custo e a facilidade de utilização são importantes, mas não são prioritários num protótipo inicial. Por isso são considerados como princípios secundários.

Em suma, a concepção da plataforma rege-se primariamente por critérios de segurança, flexibilidade e simplificação tecnológica e secundariamente por fiabilidade, autonomia, custo e facilidade de utilização.

Id.	Requisito	Quantificação
R1	Autonomia	> 2 h
R2	Velocidade máxima	> 10 km/h
R3	Velocidade mínima	< 1 km/h
R4	Condução autónoma	Acelera, desacelera e vira por comandos de um computador sem ligações exteriores à plataforma
R5	Condução manual	Acelera, desacelera e vira por ações humanas sem interface computorizada
R6	Imobilização	Desacelera até se imobilizar
R7	Imobilização	Permite bloquear imediatamente as rodas em caso de emergência
R8	Imobilização	Permite apenas desbloquear as rodas após ação humana deliberada

Tabela 2.1: Requisitos funcionais do ROVIM.

2.2 Abordagem

rever (5) O objectivo deste projeto é a construção de um protótipo funcional de um veículo autónomo destinado à vigilância de instalações militares. Na seção 2.1 foram definidos os seus requisitos funcionais, bem como os princípios a seguir no seu desenho.

O protótipo a construir pode ser visto como três atuadores embarcados num *chassis*, e respetivo *hardware* e *software* de suporte. A projeção de cada um será, exceto quando explicitamente mencionada, aproximada como independente, de modo a simplificar o dimensionamento dos componentes.

Analisando as especificidades dos atuadores, observa-se que o seu dimensionamento se assemelha a dois tipos de problemas distintos: a construção de um veículo elétrico e a construção de um robô.

A literatura existente valida esta distinção. A literatura sobre veículos elétricos é vasta, mas trata maioritariamente sobre veículos com condutor. A propulsão e armazenamento de energia são amplamente discutidos, mas outros sistemas elétricos são ignorados. A direção do veículo é melhor endereçada na literatura de robótica, por se assemelhar a um problema de controlo.

Assim, o projeto é dividido em duas áreas do saber: veículos elétricos, que compreende o *chassis*, sistema de tração e baterias, e robótica, que compreende o desenho do sistema de viragem. O

sistema de travagem não é tratado neste capítulo, pois a tecnologia usada já é endereçada para os outros dois sistemas.

2.3 Revisão da literatura

2.3.1 Propulsão

O veículo a motor é hoje em dia um componente fundamental da sociedade, e a base da plataforma Robô de Vigilância de Instalações Militares (ROVIM). Existem dois tipos principais de motores usados na propulsão de veículos: o motor de combustão interna e o motor elétrico.
O motor de combustão interna é uma tecnologia fiável e refinada ao longo de mais de um século. No entanto, é maioritariamente usada em veículos conduzidos por humanos. Adaptar um veículo com motor de combustão interna para condução autónoma exige um conjunto de alterações ao nível do seu funcionamento interno (no caso de motores mais antigos sem controlo eletrónico), da sua gestão e do acoplamento mecânico demasiado complicadas e dispendiosas em relação à adoção de motores elétricos. Estes permitem, através dos controladores eletrónicos disponíveis atualmente, controlar com exatidão a potência produzida e a direção do movimento.
A propulsão puramente elétrica é principalmente adequada para veículos pequenos, de baixa velocidade e curta autonomia [1], pelo que se adequa perfeitamente para o ROVIM. O sistema de propulsão é a parte fundamental de um veículo elétrico [1]. É composto pelo motor e o seu controlador, o acoplamento mecânico e as rodas.

2.3.2 Acoplamento mecânico e rodas

As rodas dos veículos elétricos são idênticas às dos veículos com motor de combustão, à exceção de algumas aplicações demasiado complexas para os objetivos deste projeto.
O acoplador mecânico transmite a potência do motor para as rodas. É dimensionado em conjunto com o motor, tendo em conta as necessidades de locomoção e a disposição dos componentes no veículo. Pode ser apenas um ligação solidária entre os veios do motor e o eixo das rodas, mas tipicamente consiste num conjunto de engrenagens que desmultiplica a rotação do motor, de modo a aumentar o binário disponível nas rodas. A esta peça dá-se o nome de redutor, quando a razão de desmultiplicação é fixa, e caixa de velocidades, quando é variável. Devido à maior disponibilidade de binário dos motores elétricos, um redutor é suficiente para aplicações tipicas, o que simplifica o controlo do sistema.
Há três características técnicas a ter em conta na escolha de redutores para este projeto: as dimensões, o alinhamento dos eixos e a direção da transmissão. As dimensões e o alinhamento dos eixos são importantes no planeamento da disposição dos componentes no veículo. A direção da transmissão consiste no definição do veio de entrada (onde é aplicado o binário) e de saída do redutor. Há topologias de redutores, como o parafuso sem-fim e coroa, que não permitem facilmente reverter a

direção da transmissão [2].
O redutor do motor de tração deverá transmitir binário bidirecionalmente, para permitir que o veículo possa movimentar-se em ponto morto.

2.3.3 Controlo

O controlo do sistema de propulsão de um veículo elétrico consiste dum conversor de potência e um controlador eletrónico. O conversor converte e regula a energia fornecida pelas baterias ao motor. O controlador atua sobre o conversor, implementando o algoritmo de controlo.
Existem varias topologias destes componentes, mas a sua escolha depende diretamente do motor a controlar, pelo que se torna secundária em relação à escolha do motor.

2.3.4 Motor

Existem vários tipos de motor elétrico com potencial técnico para uso na locomoção de veículos: o motor *Direct Current* (DC) com escovas, o motor *Alternating Current* (AC) de indução, o motor síncrono de ímanes permanentes, o motor síncrono de escovas, o motor de relutância variável e o motor de Lynch.
O motor de relutância variável consiste num rotor com pólos salientes de material magnético. Os rotor gira consoante o campo magnético produzido no estator, de modo a minimizar a relutância do circuito magnético. Estes são baratos, simples e com boas características de binário e velocidade para uso em veículos elétricos, mas não são conhecidas aplicações comerciais desta tecnologia [3].
O motor de indução é um motor muito simples, robusto e tecnologicamente maduro. Consiste num rotor bobinado, em que a corrente é induzida pelo campo magnético gerado nos enrolamentos do estator. Esta por sua vez gera um campo magnético que tende a seguir o do estator, gerando movimento. É especialmente indicado para aplicações estáticas, mas pode ser adaptado para aplicações de veículos elétricos, com recurso a técnicas avançadas de controlo. O motor síncrono de ímanes permanentes é um motor que gera um campo magnético constante no rotor, com recurso a ímanes permanentes. Este gira sincronamente com o campo do gerado nos enrolamentos do estator. Os motores síncronos de ímanes permanentes são alimentados por corrente AC, com recurso a controladores específicos. Este tipo de motores é muitas vezes equipado com inversores que permitem que sejam alimentados por corrente DC. Nesse caso tomam a designação de motores DC sem escovas. Devido ao uso de ímanes permanentes, estes motores conseguem atingir uma elevada eficiência e adquirir formas não convencionais.
Os ímanes permanentes nos motores síncronos podem ser substituídos por enrolamentos no rotor, alimentados por corrente DC, com recurso a escovas e anéis coletores. Estes motores têm um controlo mais simples e mais adaptável que os motores de ímanes permanentes, mas pior rendimento.
O motor DC com escovas consiste num rotor com vários enrolamentos, alimentados por um sistema de comutação com escovas, e um estator que gera um campo estático. O estator pode ser bobinado, ou de ímanes permanentes. Estes são motores fiáveis, tecnologicamente maduros, baratos e fáceis de controlar. Para além disso é possível controlar o fluxo magnético e o binário independentemente.

As suas principais desvantagens são o baixo rendimento e densidade de potência, e as necessidades de manutenção e contra-indicação para uso em ambientes inflamáveis, devido às faíscas e ao desgaste produzidos na comutação das escovas.

Uma configuração particular do motor DC com escovas de ímanes permanentes é o motor de Lynch. Em [4] é apresentada uma descrição da estrutura do motor e da sua performance. Este é um motor de fluxo magnético axial, ao contrário das restantes tecnologias apresentadas (de fluxo radial). O rotor consiste em enrolamentos laminares de cobre, com "dentes" de ferro compactados entre os enrolamentos, que ajudam a conduzir o fluxo. O estator é formado por dois discos de ímanes permanentes que o ladeiam. Este motor apresenta algumas das vantagens dos motores DC de ímanes permanentes, nomeadamente rendimento e densidade de potência elevados, mas é a sua compactidade a principal vantagem em aplicações de pequenos veículos elétricos, pois permite maior flexibilidade no desenho de soluções de propulsão. Além disso, é atualmente usado em aplicações de pequenos barcos, motas e veículos elétricos de porte semelhante ao da plataforma ROVIM.

Em [1], [3] e [5] são apresentadas comparações semelhantes dos méritos destas topologias de motores (à excepção do motor de Lynch) para soluções comerciais de veículos rodoviários. Nos três casos conclui-se que os motores síncronos de ímanes permanentes e os motores de indução são os mais indicados. Esta comparação é um excelente ponto de partida no processo de escolha do motor de propulsão da plataforma. No entanto, estas aplicações têm diferentes requisitos dos da plataforma ROVIM, pelo que se impõe uma análise critica destes resultados.

Em [5] é apresentada uma tabela (tabela II) com a avaliação dos diferentes sistemas de tração para veículos elétricos. Os critérios de avaliação (de igual ponderação) usados são: densidade de potência, eficiência, controlabilidade, fiabilidade, maturidade e custo. Tendo em conta os critérios de desenho e os requisitos da plataforma ROVIM, estes critérios foram ponderados da seguinte forma: densidade de potência, controlabilidade, fiabilidade, e maturidade valem o dobro na classificação final, enquanto que a eficiência e o custo valem o seu valor apenas. Para poder comparar o motor de Lynch com as outras topologias, este foi avaliado de forma idêntica (à luz dos requisitos desta aplicação). Foi-lhe atribuída a seguinte classificação (de 0 a 5):

Densidade de potência 5: apresenta dimensões reduzidas (especialmente o comprimento axial) que facilitam a instalação em veículos pequenos e elevada densidade de potência [4];

Eficiência 3,5: cerca de 93 % de eficiência máxima [6] (idêntica à do motor de relutância [5]);

Controlabilidade 5: igual à de um motor DC com escovas;

Fiabilidade 3: igual à de um motor DC com escovas;

Maturidade 4,5: patenteado em 1986. Desenvolvido para aplicações semelhantes ao ROVIM;

Custo 3: Custo idêntico ao de um motor síncrono de ímanes permanentes;

O motor Lynch é também usado num dos protótipo do Projeto FST Novabase, o que permite potenciais trocas de experiências e impressões entre as equipas de concepção e acesso mais fácil a

peças suplentes. Para além disso, este motor possui uma gama de controladores recomendados pela marca, o que simplifica o seu processo de escolha e aumenta o grau de confiança na solução projetada. Por isso, foram atribuídos a este motor 4 pontos adicionais à classificação final.
A tabela 2.2 apresenta a classificação das topologias apresentadas, revista de acordo com os requisitos deste projeto. Dos resultados expressos na tabela 2.2 se concluí que o motor de indução, de

Critério	peso	Topologia				
		DC	Indução	Ímanes permanentes	Relutância variável	Lynch
Densidade de potência	x2	2,5	3,5	5	3,5	5
Eficiência	x1	2,5	3,5	5	3,5	3,5
Controlabilidade	x2	5	5	5	3	5
Fiabilidade	x2	3	5	4	5	3
Maturidade	x2	5	5	4	4	4,5
Custo	x1	4	5	3	4	3
Pré-total		37,5	45,5	44	38,5	45,5
Extra		0	0	0	0	4
Total		37,5	45,5	44	38,5	45,5

Tabela 2.2: Avaliação das várias topologias de motores elétricos segundo os requisitos do ROVIM.

ímanes permanentes e o motor de Lynch são os mais adequados para satisfazer os requisitos desta aplicação, não havendo um vencedor definido. Isto significa que a escolha do motor não recai nos méritos das várias tecnologias, mas em outros aspetos práticos de implementação.

2.3.5 Armazenamento de energia

A fonte de energia dos veículos puramente elétricos é o principal entrave à sua massificação [1]. As principais tecnologias usadas atualmente para acumular energia em veículos puramente elétricos são: super condensadores, baterias elétricas e células de combustível.
A célula de combustível é um dispositivo eletroquímico que usa uma reação química para gerar energia elétrica. Esta reação não é reversível na célula. Esta tecnologia não é ainda considerada comercialmente viável [1].
Os super condensadores são condensadores de capacidade muito elevada. Têm uma densidade de energia (kWh/Kg) extremamente baixa [1], mas uma densidade de potência (kW/kg) bastante elevada. Por isso podem ser usados como fonte auxiliar de energia para picos de potência. Em combinação com baterias tradicionais, permitem aumentar a autonomia e reduzir o desgaste das baterias. No entanto, esta combinação introduz complexidade adicional à gestão da baterias.
As baterias elétricas são atualmente a principal fonte de energia dos veículos puramente elétricos [1].

Estas transformam energia química acumulada em energia elétrica. Nas baterias usadas em veículos elétricos a reação química que produz a energia elétrica é reversível, o que permite recarregá-las e reutilizá-las.

Em [7] são apresentadas as tecnologias de baterias mais comuns para uso em veículos elétricos. Estas dividem-se em três famílias principais: ácido-chumbo, à base de lítio e à base de níquel. Destas três famílias, três tecnologias merecem especial destaque pela sua adequação ao uso em veículos elétricos: *Valve-Regulated Lead-Acid* (VRLA), *Nickel Metal Hydride* (NiMH) e *Lithium-ion* (Li-Ion).

As baterias de iões de lítio possuem a melhor densidade de energia e excelente densidade de potência. São usadas maioritariamente em aplicações de baixa potência, mas estão em constante evolução tecnológica e a sua adopção tem aumentado gradualmente. No entanto são mais instáveis, caras, e são suscetíveis a incendiar-se em caso de sobrecarga.

As baterias de NiMH são a tecnologia mais usada em veículos elétricos [7]. Possuem elevada densidade de potência e são bastante seguras de operar.

As baterias de VRLA são baterias de ácido e chumbo, uma tecnologia extremamente robusta e disseminada, mas seladas e com regulação por válvula, que as tornam mais seguras e tolerantes a sobrecarga. Possuem uma boa densidade de potência e são baratas, mas têm fraca densidade de energia, o que as torna pouco adequadas para aplicações de baixo peso e/ou elevada autonomia. No entanto, devido à sua simplicidade e robustez, são válidas para uso em protótipos; em particular para uso nesta fase do projeto ROVIM, que requer baixa velocidade e curta autonomia.

Aplicações com várias baterias ligadas em série, como é o caso das aplicações em veículos elétricos requererem uma unidade de monitorização e balanceamento de cargas e temperatura. Isto é especialmente verdade para as baterias de NiMH e Li-Ion. As baterias de VRLA suportam melhor temperaturas adversas.

2.3.6 Direção

rever (7) **rever** (8) O sistema de direção do ROVIM deve permitir controlar o ângulo de viragem do veículo por computador.

A direcção de uma moto-quatro é tipicamente um sistema de Ackermann atuado através de um guiador. Esta é uma geometria de viragem coordenada de duas rodas paralelas, que permite mantê-las num movimento de rotação pura (sem derrapagem) durante uma viragem em que estas seguem trajetórias de raios diferentes.

Este sistema pode ser controlado por um controlador Proporcional Integral Derivativo (PID) simples. O controlador é composto pelos componentes: o atuador (o motor e o acoplador mecânico), o dispositivo de retroalimentação (um sensor de posição) e um controlador que execute o algoritmo de controlo.**To do** (9)**To do** (10)

Para projetar um controlador para o sistema, este precisa de ser traduzido por um modelo matemático. Existem duas abordagens para o fazer: deduzir as equações do movimento do sistema, ou estimá-las usando um método de identificação.

To do (11)

2.4 Planeamento

A execução do projeto foi planeada numa sequência natural de planeamento, construção e validação, de modo a minimizar o custo de alterações futuras. Para validar a execução foi definido um objetivo preliminar de instalação dos componentes principais no veículo e um objetivo intermédio: a implementação das funcionalidades de condução manual.

Após instalar os componentes principais (os acumuladores de energia, atuadores e acoplamentos mecânicos) no veículo, as necessidades logísticas e de recursos difíceis de obter num laboratório de eletrónica diminuem.

A funcionalidade de condução manual do veículo é um objetivo mais simples que o objetivo final e permite validar a execução do projeto numa etapa intermédia, e continuar de seguida em direção ao objetivo final sem necessidade de executar modificações relevantes no veículo.

Assim, inicialmente cada subsistema foi projetado e dimensionado com o detalhe que a visão inicial sobre o projeto permitiu. Foi possível compreender as suas características fundamentais, mas pormenores avançados de implementação foram capturados e resolvidos nas fases de construção e validação.

De seguida foram adquiridos e acomodados no *chassis* os componentes principais, de modo a validar o planeamento da sua disposição e atingir o objetivo preliminar.

A terceira fase consistiu na instalação das funcionalidades de tração e travagem e dos sistemas de segurança, de modo a atingir o objetivo intermédio do projeto.

Por fim partiu-se para o objetivo final, que a este ponto consistia no controlo da direção e integração das várias funcionalidades e o seu comando por computador.

3

Componentes do sistema

Descrever o resultado final, subsistema a subsistema (em cada secção é apresentado um subsistema que funciona individualmente, mas não está integrado com os restantes). Referir as opções de design feitas, o dimensionamento feito.

3.1 Chassis

Dizer que a primeira coisa que se fez foi comprar o chassis, e depois escolher e colocar os componentes. Dizer porque se comprou o chassis e o que vinha e não vinha (carretos da corrente, rodas, etc). Descrever brevemente o chasis (travão de tambor, direção ackermann, eixo traseiro com cremalheira já incorporada, coluna de direção, espaço entre a forquilha traseira e parte de cima) Descrever as várias modificações, e o porquê. Dizer que foram usados parafusos onde possível. Entrar aqui com o layout dos componentes principais de forma ligeira.
imagens e tabelas: fotos do chassis despido, modificações de cada subsistema (4 fotos).

3.2 baterias

Descrever as baterias, e a sua disposição na moto. Justificar a sua escolha. Dizer que só foi possível assim por causa do motor. Apresentar um valor esperado para a autonomia (com base nos cálculos do motor de lynch), mas dizer que se incluíram o máximo que se conseguiu no espaço que havia. Dizer que se optou por ir à campeão, sem sistema de gestão das baterias. Referir as limitações no isolamento dos circuitos e no facto de não estarem fixas à moto. Referir os carregadores.
imagens e tabelas: vista expandida das baterias, com os espaçadores..

3.3 traçao

Dizer qual o motor que se escolheu, e porquê. Descrever o kit que se comprou. Apresentar cálculos básicos das necessidades de potência, mas dizer que a escolha não se regeu por isso, mas sim pelo kit facilitar a escolha e compra do resto. Dizer que se procuraram redutores, mas achámos ou muito caros, ou com pouca relação de redução, ou unidirecionais, o que não pode ser para aqui pq queremos andar com a moto desengatada.
Dizer que se fez um redutor, e descrevê-lo. Apresentar os esquemas e a errata.
Apresentar o controlador e as suas ligações (modo manual). Apresentar a sua configuração. Dizer que este está ligado a um sistema de segurança integrada, e apontar para a secção.
Apresentar o sensor de velocidade: feito para feedback ao controlador e para o cérebro. Dizer pq se escolheu este tipo. Apresentar as suas limitações para velocidade baixa. Dizer que foi instalado ali por simplicidade, mas que deveria ter sido instalado no veio do motor, ou idealmente escolhida outra topologia. Referir apenas os problemas de leitura e o condicionamento de sinal na secçào do hw de controlo, pq só foram feitos por causa do uC (e não por causa do Sigmadrive)
Apresentar os esquemas elétricos:sensor velocidade: ligaçoes controlador sem dispositivos de segurança.
imagens e tabelas: motor + redutor instalados na moto. placa do sensor. Sensor final instalado, montagem do controlador + fusível + contactor.

3.4 travagem

Descrever o que foi feito para atuar o travão: parafuso e pivô no tambor, fdc para controlar a paragem do motor. Dizer o motor escolhido e o controlador. Explicar a filosofia de travão de emergência (tudo ou nada). Apresentar as ligações do motor do travão, com fdc, sem dispositivos de segurança. Referir que o sistema é comandado por um sistema de segurança integrada, que neste caso é também o controlador do motor e apontar para a secção. imagens e tabelas: motor + ligação ao parafuso. Ligação do parafuso ao tambor.

3.5 Circuitos de segurança

Explicar a filosofia de funcionamento dos circuitos: default off, ao estilo watchdog; atuação complementar do travão e dos outros motores; minimizar e tornar robusto hw travão (relés em vez de transistores; motor ligado diretamente à bateria).
Explicar os vários botões de segurança. Referir o controlo por sw. Explicar o monoestável. Referir que este circuito só afeta os motores, pois são o que pode causar originar colisões, e que tem total prioridade de controlo sobre eles, mas que não atua no computador (não o desliga).
Referir os fdc como detetores de erros para a viragem, e como parte do sistema de controlo em cadeia aberta para o travão. Explicar os mecanismos de destravagem. Explicar que é o sistema

está preparado para destravar para o sw, mas como o acionamento do travão é considerado um erro grave, que tem que se feito manualmente. No futuro pode ser alterado. Apontar nas imagens do tambor do travão e do potenciometro das secções anteriores os fdc. Apresentar esquema elétrico dos controlos de segurança (sem motores). Apontar para imagem da caixa da eletrónica e referir a placa dos relés.
imagens e tabelas: disp. homem morto.

3.6 direção

Descrever como se ligou o atuador ao sistema manual de direção. Descrever o motor, redutor (e porque aqui dá jeito um redutor de pinhão e coroa) e engrenagens. Descrever que a correia tem um esticador para poder ser retirado para condução manual (virar com o guiador). Descrever o conversor e a sua ligação à bateria. Descrever o sistema de feedback, e como ele foi instalado na moto. Descrever o controlador e porquê se obtou por ele. Falar por alto que se usou um pic para implementar o algoritmo de controlo e servir de controlador geral do sistema. Indicar a secção onde ele está descrito.

Apontar para a imagem da caixa de eletrónica e referir a placa com o contactor e o fusível.
images e tabelas: motor + redutor + correia e engrenagens instalados na moto, potenciometro, fdc e roda dentada instalados na moto, esquema elétrico do sensor e da alimentação do motor.

3.7 Integração dos sistemas

Descrever a possibilidade de condução por manual e por computador. Dizer como isto foi feito: os sinais de controlo para o controlador da tração foram re-roteados por seletores do modo de condução, de forma a se poder injetar sinal do computador. O travão manteve o seu modo de funcionamento, só foi adicionada uma forma de comando adicional por sw. O controlador da direção só está previsto funcionar por sw. Dizer que o sistema de segurança tem prioridade na atuação nos motores, mas que não interfere na atuação normal do controlador.
Apresentar o uc. Referir as suas funções: monitorização do sistema (medição de sinais), interface com o módulo superior, controlo do conversor do travão e dos controladores da tração e travão.
Apresentar os sinais que interagem com o uC, o seu condicionamento. Apresentar os interruptores e leds. Apresentar a condução manual. Referir o que não pode ser controlado por computador: ligar/desligar e recuperar de erros graves (acionar o travão).
Apresentar a condução por computador: comandada por sw (apontar para a secção), e só consegue desacelerar usando a travagem regenerativa. Às velocidades previstas para a moto não se prevêm dificuldades de controlo. O controlador da tração foi configurado para deixar deslizar a moto quando está em neutro, e para manter a posição quando está selecionada uma direção de movimento. imagens e tabelas: caixa da eletrónica; tabela de elementos de interface com o utilizador (inputs e outputs).

3.8 controlo da direção

Dizer que se planeou um pid originalmente. Identificar as caracteristicas de operação: terreno variado => grande variação do coeficiente de atrito e possível força lateral de perturbação, quando se tenta virar em cima de pedras, que apesar do redutor usado, ainda consegue alterar a posição da direção ou aumentar muitoa resistência ao movimento => muitas e fortes perturabações no sistema de controlo. Para além disso, à medida que se chega aos fdc, a força necessária para virar aumenta, o que significa que este é um sistema não-linear.No fim da instalação do atuador foram introduzidos dois problemas adicionais difíceis de resolver: a zona morta na atuação da direção, e o salto da correia quando esta no chão de azulejo (baixo atrito - melhor cenário). Obtou-se por avançar com essas limitações em vez de as corrigir. Isso implica que a moto só possa virar quando está em movimento.
Explicar o modelo do sistema criado. Referir que se optou por usar o mecanismo de controlo de motores em cadeia fechada do dalf, e dizer como funciona.
Aproximou-se o sistema a um slit, e tentou-se modelar o sistema com recurso à identificação pelo método de ziegler nichols, mas os resultados não foram satisfatórios, pelo que se partiu para a afinação a olho dos parametros do PID, para o caso das rodas no ar.
Apresentar o PID. Referir que este é um controlo de prova de conceito, mas com bastantes limitações. Mais testes de refinação poderiam produzir uma coisa melhor. Referir que se quisesse fazer um PID melhor com o que tenho, arranjava um PID para cada tipo de terreno e arranjava uma opção de seleção pelo utilizador.
imagens e tabelas: diagrama de blocos do pid ideal: zona morta + pid diferente consoante o tipo de terreno + perturbação no sinal de comando devido ao saltar da correia + perturabação na variável controlada devido a irregularidades do terreno (ruído no sensor não é relevante); diagrama com a representação do ângulo da direção (0°, 45° bombordo estibordo).

3.9 Programa de controlo

Apresentar e explicar o diagrama de estados do programa. Explicar o estado de lockdown. Explicar a arquitetura do programa de controlo: escuta e execução de comandos, e daemons de controlo dos motores(direção usei o do dalf, e tração criei eu o pwm) e monitorização do sistema. Explicar a arquitetura do firmware dalf. Explicar como foi construído: o programa foi construído em cima do firmware dalf, de duas maneiras: adicões à funcionalidade da plataforma, que não são específicas para o rovim, e software destinado apenas ao rovim, de implementação do programa de controlo, mas que precisava de funcionalidades genéricas que a plataforma não tinha. imagens e tabelas: diagrama de estados. diagrama de threads.

4

Funcionamento do ROVIM

as secções das funcionalidades são apenas os fluxogramas.

4.1 Ligar

4.2 Desligar

4.3 Selecionar/desseleccionar automático

4.4 Acelerar

4.5 Desacelerar

4.6 Virar

4.7 travagem de emergência/Ir para lockdown

4.8 Sair de lockdown

5

Conclusão

Objetivo Resumo da tese para quem a leu.

Topicos de escrita

- especificações da plataforma
- Limitações da plataforma
- Comparar as especificações com os objetivos iniciais

Notas a escrever O objetivo principal, de produzir um protótipo funcional foi atingido, ainda que com algumas limitações, aceitáveis para esta fase do processo de prototipagem e produção.

Limitações de ordem logística e de equipamento (ferramenta disponível, experiência, material disponível para compra) tiveram impacto considerável no desenvolvimento do projeto. Conlui-se daqui que num projeto prático deste tipo as considerações tecnológicas representam apenas parte dos constrangimentos. Para as baterias não sei como evitar que se mexam.

Para a zona morta, soldar a coluna da direção em vez de aparafusar.

Para o salto da correia, usar uma corrente (talvez de bicicleta), com alguma folga, e uma "chave"do estilo da usada para prender a engrenagem ao veio do motor, facilmente removível, para resolver o problema do salto.

O hardware parece bastante precário, especialmente ao nível das placas elétronicas, mas também das baterias, mas pode ser usado com confiança por utilizadores conhecedores da plataforma. Para isso foi produzido um manual do utilizador.

5.1 Trabalhos futuros

Refinação do protótipo. Dois caminhos de melhoramento: refinação do hardware atual, e construção de um novo protótipo com as lições aprendidas.

Refinação do hw atual:

passo seguinte, por este veículo ainda ter um potencial de refinamento muito grande.

Testes: desenhar um programa de testes de aceitação compreensivo, que premita aumentar o grau de confiança no veículo e caracterizar melhor as propriedades do veículo. Usar estes testes para identificar áreas de melhoria na construção de um novo veículo.

Integrar os módulos superiores. Novo protótipo:

corrigir erros.

Integrar requisitos de elevada fiabilidade, autonomia, custo baixo, e reprodutibilidade (standardização de peças, definição de medidas).

Poupar dinheiro, redimensionando componentes (motores, controlador de tração). Ponderar o estado de novas tecnologias de baterias para aumentar a capacidade.

Bibliografia

[1] C. C. Chan, “The state of the art of electric and hybrid vehicles [prolog],” *Proceedings of the IEEE*, vol. 90, no. 2, pp. 245–246, Feb 2002.

[2] “Introduction to gears,” guia, KOHARA GEAR INDUSTRY CO., LTD., 2006.

[3] J. de Santiago, H. Bernhoff, B. Ekergård, S. Eriksson, S. Ferhatovic, R. Waters, and M. Leijon, “Electrical motor drivelines in commercial all-electric vehicles: A review,” *IEEE Transactions on Vehicular Technology*, vol. 61, no. 2, pp. 475–484, Feb 2012.

[4] S. M. A. Sharkh and M. T. N. Mohammad, “Axial field permanent magnet dc motor with powder iron armature,” *IEEE Transactions on Energy Conversion*, vol. 22, no. 3, pp. 608–613, Sept 2007.

[5] N. Hashemnia and B. Asaei, “Comparative study of using different electric motors in the electric vehicles,” in *Electrical Machines, 2008. ICEM 2008. 18th International Conference on*, Sept 2008, pp. 1–5.

[6] (2016) página *web*. Saietta Group. [Online]. Available: http://www.saiettaengineering.com/motors/

[7] K. Yiu, “Battery technologies for electric vehicles and other green industrial projects,” in *Power Electronics Systems and Applications (PESA), 2011 4th International Conference on*, June 2011, pp. 1–2.

[8] B. Shamah, M. D. Wagner , S. Moorehead, J. Teza, D. Wettergreen, and W. R. L. Whittaker, “Steering and control of a passively articulated robot,” in *SPIE, Sensor Fusion and Decentralized Control in Robotic Systems IV*, vol. 4571, October 2001.

[9] “Modelação e simulação,” slides das aulas teóricas, 2008.

[10] “Controlo,” transparências de apoio às aulas teóricas, 2007.

[11] K. Rajashekara, “History of electric vehicles in general motors,” in *Industry Applications Society Annual Meeting, 1993., Conference Record of the 1993 IEEE*, Oct 1993, pp. 447–454 vol.1.

[12] *Saietta Motors Workshop Manual*, Saietta Group, 2015. [Online]. Available: http://www.saiettaengineering.com/se_website/wp-content/uploads/2015/11/Saietta-Motors-Workshop-Manual_Nov-2015.pdf

[13] “Agni motor thermistor information,” nota técnica, EV Power.

[14] “Genesis NP range overview,” brochura, EnerSys Inc., Feb. 2005.

[15] “Genesis NP55-12/NP55-12FR,” ficha técnica, EnerSys Inc., May 2003.

[16] “Standard metric keys and keyways for metric bores with one key • couplings,” tabelas com as dimensões da norma ISO/R773, The Falk Corporation, Jul. 1996.

[17] Catálogo de engrenagens, Eurocorreias, Lda., 2012.

[18] “Bearings specification tables,” catálogo, JTEKT Corporation, 2009.

[19] “Sigmadrive pm traction technical manual,” PG Drives Technology, 2008.

[20] “Rs dc servomotors,” manual técnico, PARVEX SAS, 2003.

To do (12)

Apêndice A - código do programa

O código fonte usado na programação da placa de controlo do T2D é aqui listado. Uma cópia exata destes ficheiros, assim como outros ficheiros usados no projeto está acessível em: https://github.com/ROVIM-T2D/ROVIM-T2D-Brain/tree/v1.0.

Os ficheiros `main.c` e `dalf.h` são alterações aos ficheiros originais fornecidos com a placa Dalf para esta aplicação, e são licenciados pela *Embedded Electronics, LLC*.

A biblioteca `dalf.lib` é também necessária para gerar o programa, mas não é listada aqui por não ter sido alterada da versão original, por estar em formato binário, e por regras de licensiamento restritas. A licensa destes ficheiros pode ser consultada no site do fornecedor, em: http://www.embeddedelectronics.net/documents/Ver160/EULA.pdf.

To do (13)

Fix (14)

```
1  #line 1 "main.c"           //work around the __FILE__ screwup on windows, http://www.
       microchip.com/forums/m746272.aspx
2  //cannot set breakpoints if this directive is used:
3  //info: http://www.microchip.com/forums/m105540-print.aspx
4  //uncomment only when breakpoints are no longer needed
5  /*********************************************************************
6  *********************************************************************
7  **
8  **
9  **                    main.c
10 **
11 **       This is the main.c file of the Dalf-1F v1.73 firmware, modified
12 **       for the ROVIM project.
13 **
14 **       See Dalf-1F owner's manual and the ROVIM T2D documentation for more
15 **       details.
16 **
17 **           The ROVIM Project
18 **
19 **       This is a derivation from a file provided by Embedded Electronics, LLC
20 *********************************************************************
21 *********************************************************************/
22
23 /*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
24 ;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
25 ;xx                                                                     xx
26 ;xx           M       M        AAAAA       IIIIIII      N     N        xx
27 ;xx           M M   M M       A     A        I         N N   N        xx
28 ;xx           M   M   M       AAAAAAA        I         N  N  N        xx
29 ;xx           M       M       A     A        I         N   N N        xx
30 ;xx           M       M       A     A     IIIIIII      N     N        xx
31 ;xx                                                                     xx
32 ;xx     ================================================================    xx
33 ;xx     This is the C Language "backbone" for the Dalf Motor Application.    xx
34 ;xx     The code shown here includes: main loop, some power up init,         xx
35 ;xx     interrupt service dispatch, and service requests initiated by the    xx
36 ;xx     various interrupt service handlers.                                  xx
37 ;xx                                                                          xx
38 ;xx     The low level code (interrupt services, PID, Trajectory Generator,   xx
39 ;xx     C library routines, various utilities) is written in PIC Assembly    xx
40 ;xx     with the actual code located in the <dalf.lib> file.                 xx
41 ;xx     ================================================================    xx
42 ;xx                                                                          xx
43 ;xx                                                                          xx
44 ;xx                          Microchip tools                                 xx
45 ;xx     +-------------------------------------------------------------+      xx
46 ;xx     |  Tool         Ver        Description                        |      xx
47 ;xx     |  ---------    -----   ------------------------------------  |      xx
48 ;xx     |  MPLAB IDE    8.00    Integrated Development Environment    |      xx
49 ;xx     |  MCC18        3.10    C Compiler for the PIC18F device family|     xx
50 ;xx     |  MPASM        5.10    Assembler                             |      xx
```

```
;xx     |  MPLINK     4.10    Linker                                          |     xx
;xx     |  MPLIB      4.10    Librarian                                       |     xx
;xx     +-------------------------------------------------------------------+     xx
;xx                                                                               xx
;xx                              -H I S T O R Y-                                  xx
;xx                                                                               xx
;xx     DATE     WHO       DESCRIPTION                                           xx
;xx  ---------  ---  ---------------------------------------------------------   xx
;xx  09/15/06  SMB  Ver 1.40.  First production code release.                     xx
;xx  11/30/06  SMB  Ver 1.50.  New Servo modes, analog feedback, et. al.         xx
;xx  06/08/07  SMB  Ver 1.60.  Serial comm fix.                                  xx
;xx  06/28/07  SMB  Ver 1.62.  Beta for 1.70 Release.  Improved sync             xx
;xx  08/07/07  SMB  Ver 1.70.  New PotMix mode. First GUI compatible ver.        xx
;xx  11/05/07  SMB  Ver 1.71.  Minor changes to accomodate latest Microchip      xx
;xx                            tools (required for use with Version 1.71         xx
;xx                            development files).                               xx
;xx                                                                               xx
;xx   FILE: <main.c>                                                             xx
;xx                                                                               xx
;xx  (c) Copyright 2006 Embedded Electronics LLC, All rights reserved            xx
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
*/
#include    "p18f6722.h"                        /* Microcontroller specific definitions */
#include    "config.h"                          /* Fuse Settings */
#include    "dalf.h"
#include    <stdio.h>

//choose the configuration profile
#include    "rovim_t2d.h"                           /* description and configuration of the
    ROVIM system */

#ifdef DALF_TEST_ENABLED
    #include    "dalf_test.h"                           /* testing and debugging features */
#endif

/* Function Prototypes */
void    ISRHI ( void );
void    ISRLO ( void );
#ifdef DALF_ROVIM_T2D
void    INT1_ISR_SINGLE_SOURCE (void);
#endif //DALF_ROVIM_T2D
//void  Greeting(void);             // Terminal emulator greeting

    // Command Handling
void    SerialCmdDispatch(void);    // USART1 or I2C2 control
void    PotOrRc(BYTE mtr);          // Pot or RC control
void    PotCcmd(BYTE mtr);          // Cmd: PotC
void    PotFcmd(BYTE mtr);          // Cmd: PotF
void    PotServoCmd(BYTE mtr);      // Cmd: PotServo
void    PotMixCmd(void);            // Cmd: Pot Mix
void    RcCmd(BYTE mtr);            // Cmd: R/C Normal
void    RcMixCmd(void);             // Cmd: R/C Mix
void    RcServoCmd(BYTE mtr);       // Cmd: R/C Servo
void    CheckCmdTimeout(void);      // Check for Cmd Interface Timeout

    // Conversion
BYTE    PotCtoPWM(BYTE adc);        // Map PotC position to PWM (0-100%)
BYTE    PotFtoPWM(BYTE adc);        // Map PotF position to PWM (0-100%)
                                    // Pulse width to +/-PWM [-100, 100]
int     RCtoPWM(WORD p, WORD rcmin, WORD rcmax, WORD rcm);
//WORD  AdcConvert(WORD Adc);       // ADC reading to millivolts
WORD    PulseConvert(WORD Pulse);   // PulseWidth ticks to microseconds
WORD    AdcToVbatt(WORD v6);        // AN6 mV to Vbatt mV

    // Move Motor
//void  SoftStop(BYTE mtr);         // Stop Motor, leaving mode unchanged.
//void  MoveMtrOpenLoop(BYTE mtr, BYTE dir, BYTE spd, BYTE slew);
//void  MoveMtrClosedLoop(BYTE mtr, short long tgt, WORD v, WORD a);

    // Output
//int       printf(const rom char *fmt, ...);
void    ReturnData(void);       // Return data and/or status to cmd initiator.
```

```
123 void     DispCHR(void);            // xCHR
124 void     DispE(void);              // Motor Position (encoder)
125 void     DispV(void);              // Motor Velocity
126 void     DispADC(void);            // ADC Snapshot (AN0 .. AN6)
127 void     DispRC(void);             // RC Snapshot (Ch0 .. Ch2)
128 void     DispMEM(void);            // Single Memory Byte.
129 void     DispIO(void);             // Single IOEXP register.
130 void     DispSTAT(void);           // Motor Status
131 void     MtrStatTE(BYTE mtr);      // Motor Status Utility for TE
132 void     DispPID(void);            // Motor PID runtime parameters
133 void     DispHMS(void);            // HH:MM:SS (RTC Time)
134 void     DispExtEEBLOCK(void);     // Block of External EEPROM starting at 'pBYTE'
135 void     DispRAMBLOCK(void);       // Block of RAM starting at 'pBYTE'
136 void     DispIntEEBLOCK(void);     // Block of Internal EEPROM starting at 'pBYTE'
137 void     Tune1(void);              // Mtr1 PID Tuning Aid: CmdQ Verbose output
138 void     Tune2(void);              // Mtr2 PID Tuning Aid: CmdQ Verbose output
139
140     // Miscellaneous
141 void     ResetPIC(void);           // Reset Dalf Board
142 void     WinkLEDS(void);           // Briefly blink programmable LED's
143 void     InitLED(void);            // Initialization for on-board LED's
144 void     ServiceLED(void);         // Periodic LED service
145
146     // Over Current
147 void     Motor1Current(void);      // Voltage Window Transition - OverCurrent
148 void     Motor2Current(void);      // Voltage Window Transition - OverCurrent
149
150     // Communication
151 void     GetApiPktCkSum(void);     // Compute and store Api Pkt CkSum byte
152 void     GetI2C2PktCkSum(void);    // Compute and store I2C2 Pkt CkSum Byte
153 void     SendApiPkt(void);         // Transmit API Pkt to Host
154 void     SendI2C2Pkt(void);        // Transmit I2C2 Pkt[] to Host
155 void     WriteSSP2BUF(BYTE chr); // Load I2C2 Transmit Buffer
156
157
158
159
160
161
162 // ***************************************************************************
163 // **              ERAM: Runtime environment copy of Parameter Block        **
164 // ***************************************************************************
165 extern BYTE fPWM, AD_ACQ, AD_CNV, AD_GAP, SYSMODE;
166 extern BYTE X1_IODIRA, X1_IOPOLA, X1_GPINTENA, X1_DEFVALA, X1_INTCONA;
167 extern BYTE X1_IOCON, X1_GPPUA, X1_GPIOA, X1_OLATA;
168 extern BYTE X1_IODIRB, X1_IOPOLB, X1_GPINTENB, X1_DEFVALB, X1_INTCONB;
169 extern BYTE X1_GPPUB, X1_GPIOB, X1_OLATB;
170 extern BYTE X2_IODIRA, X2_IOPOLA, X2_GPINTENA, X2_DEFVALA, X2_INTCONA;
171 extern BYTE X2_IOCON, X2_GPPUA, X2_GPIOA, X2_OLATA;
172 extern BYTE X2_IODIRB, X2_IOPOLB, X2_GPINTENB, X2_DEFVALB, X2_INTCONB;
173 extern BYTE X2_GPPUB, X2_GPIOB, X2_OLATB;
174
175 extern WORD VBCAL;
176 extern WORD VBWARN;
177
178 extern BYTE AMINP;
179 extern WORD MAXERR, MAXSUM;
180
181 extern BYTE MTR1_MODE1, MTR1_MODE2, MTR1_MODE3;
182 extern WORD ACC1, VMID1;
183 //extern BYTE VSP1;
184 extern WORD KP1, KI1, KD1;
185 //extern BYTE VMIN1, VMAX1;
186 //extern WORD TPR1;
187 extern WORD MIN1, MAX1;
188
189 // extern BYTE  MTR2_MODE1, MTR2_MODE2, MTR2_MODE3;
190 //extern WORD ACC2, VMID2;
191 //extern BYTE VSP2;
192 extern WORD KP2, KI2, KD2;
193 extern BYTE VMIN2, VMAX2;
194 extern WORD TPR2;
195 extern WORD MIN2, MAX2;
```

```
196
197 extern WORD RC1MIN, RC1MAX, RC2MIN, RC2MAX, RC3MIN, RC3MAX;
198 extern BYTE RCD;
199
200 extern BYTE POT1A, POT1B, POT2A, POT2B;
201
202 extern BYTE nBR, NID, RX1TO;
203 extern WORD NPID;
204 extern BYTE Dev_DALF;
205
206 extern BYTE RCSP;
207 extern BYTE PSP;
208 extern BYTE DMAX;
209
210 extern BYTE FENBL, DECAY;
211
212 extern BYTE CMDSP, CMDTIME;
213
214 extern BYTE ERAM7A;       //UNUSED//
215 extern BYTE ERAM7B;       //UNUSED//
216 extern BYTE ERAM7C;       //UNUSED//
217 extern BYTE ERAM7D;       //UNUSED//
218 extern BYTE ERAM7E;       //UNUSED//
219 extern BYTE ERAM7F;       //UNUSED//
220 //-----------------------------------------------------------------
221
222
223
224
225
226
227 //**************************************
228 //**    Other Assembler RAM Variables    **
229 //**************************************
230 extern  BYTE    PIC_DEVID1, PIC_DEVID2;     // PIC Revision
231 extern  BYTE    BOARD_ID;                   // Board ID char
232 extern  BYTE    MAJOR_ID;                   // Software major ID byte
233 extern  BYTE    MINOR_ID;                   // Software minor ID byte
234 extern  WORD    USERID;                     // PIC_USERID[3..0] concatenated
235 //extern    BYTE    SCFG;                       // Serial Configuration (1..3)
236
237 extern  WORD    SERVICE, SERVICE_REQ;       // Requests from ISR's
238 extern  BYTE    TIMESVC, TIMESVC_REQ;       // Timed requests
239 //extern    BYTE    SECS, MINS, HOURS;      // RTC variables
240 //extern    ULONG   Seconds;                // Seconds since boot
241 //extern  BYTE    ADC0[7];                  // ADC readings AN0..AN6
242 extern  BYTE    Pkt[134];                   // Max size (API) N+6 = 128+6 = 134
243 extern  WORD    DispSvc;                    // "Display" service requests
244 extern  BYTE    DataReq;                    // Arg used in servicing printf
245 extern  BYTE    xCHR;                       // Arg used in servicing printf
246 extern  BYTE    BlkLen;                     // Arg used in Memory Blk Read
247 extern  WORD    PulseWidth1;                // EX0 (units: 1.28 uSec)
248 extern  WORD    PulseWidth2;                // EX1 (units: 1.28 uSec)
249 extern  WORD    PulseWidth3;                // EX3 (Units: 1.28 uSec)
250 extern  BYTE    NewPulse;                   // R/C Signal loss detect bits
251 extern  WORD    RC1m,RC2m,RC3m;             // Slope: PWM% vs PWidth
252 extern  WORD    RC1M,RC2M;                  // Slope: ServoPos'n vs PWidth
253 extern  WORD    RC1center;
254 extern  WORD    RC2center;
255 extern  WORD    RC3center;                  // Pulse range center (uSec)
256
257 extern  WORD    Pid1Limit;                  // Motor1 Tuning Output Control
258 extern  WORD    npid1;                      // Motor1 PID Tuning Output count
259 extern  BYTE    Mtr1_Flags1;                // Motor1 flags1
260 extern  BYTE    Mtr1_Flags2;                // See dalf.h
261 //extern  BYTE    S1,Power1;                // SPD: [0..100%], [0..VMAX1%]
262 //extern  short long  encode1;              // Mtr1 position encoder
263 //extern  short long  V1;                   // Mtr1 Velocity
264 extern  short long  x1pos;                  // Motor1 PID Target
265 extern  short long  Err1;                   // Motor1 PID Err
266 extern  short long  ErrDiff1;               // Motor1 PID Err Diff (dErr/dT)
267 extern  short long  ErrSum1;                // Motor1 PID Err Sum
268
```

```
269 extern  WORD    Pid2Limit;                  // Motor2 Tuning Output Control
270 extern  WORD    npid2;                      // Motor2 PID Tuning: Output count.
271 //extern  BYTE    Mtr2_Flags1;             // Motor2 flags1
272 //extern    BYTE    Mtr2_Flags2;           // See dalf.h
273 //extern  BYTE    S2,Power2;                 // SPD: [0..100%], [0..VMAX2%]
274 extern  short long  encode2;               // Mtr2 position encoder
275 //extern  short long  V2;                  // Mtr2 Velocity
276 extern  short long  x2pos;                 // Motor2 PID Targets
277 extern  short long  Err2;                  // Motor2 PID Err
278 extern  short long  ErrDiff2;              // Motor2 PID Err Diff (dErr/dT)
279 extern  short long  ErrSum2;               // Motor2 PID Err Sum
280
281
282 //extern    BYTE    CMD,ARG[16],ARGN;
283 extern  BYTE    ReturnN;                   // Commmand Interface
284 //extern    BYTE    CmdSource;
285
286 //extern  BYTE    LedErr;                    // "1" bits flag err cond'n
287
288 extern  BYTE    xbyte;                     // printf() arg - Hex Byte
289 extern  BYTE    ISR_Flags;                 // Flags set/cleared by ISR
290 extern  BYTE    I2C2_PktStatus;            // Flags for I2C2 Interface
291 extern  BYTE    I2C2_State;                // SLAVE ISR State.
292
293 extern  BYTE    Cmd_Ticks;                 // Cmd Interface Timeout ticker
294
295
296 //**************************************
297 //**          Other Variables           **
298 //**************************************
299
300 #ifdef WATCHDOG_ENABLED
301     WORD watchdogcount = WATCHDOG_PERIOD;
302 #endif
303
304 //**********************
305 //  Global Variables  **
306 //**********************
307 WORD    RC[3]={1500,1500,1500}; // R/C pulse widths in microseconds.
308 int     pwm1,pwm2;              // R/C pwm conversions.
309
310 BYTE    *pBYTE;                 // Memory pointer
311 BYTE    serialcmd = FALSE;      // Flag: serial interface cmd received.
312 BYTE    PktLen;                 // Length of Pkt.
313
314 // LED variables
315 ULONG   ledshift;       // On/Off LED bit shifter (shifted bit is time period)
316 ULONG   grn1pattern;    // LED1: On/Off LED bit pattern
317 ULONG   grn2pattern;    // LED2: On/Off LED bit pattern
318 ULONG   redpattern;     // LED3: On/Off LED bit pattern
319 BYTE    ledcount;       // LED Service Counter
320
321 // TIME variables
322 TIME    Delay1;
323
324 // Motor Control Modes
325 enum MotorModes
326 {
327     POT_CENTER_OFF,     //0 PotC: [-100%, 100%] with center off.
328     POT_WITH_SWITCH,    //1 PotF: [0%, 100%] with separate direction switch.
329     RADIO_NORMAL,       //2 R/C Interface, Normal (TANK) mode.
330     RADIO_MIX,          //3 R/C Interface, Mix Mode.
331     RADIO_SERVO,        //4 R/C Servo Interface.
332     POT_SERVO,          //5 POT Servo Interface.
333     POT_MIX,            //6 POT Mix Mode.
334     RS232               //7 Serial cmd interface.
335 };
336
337
338 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
339 #pragma code INTHI = 0x808
340 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
341 //-------------------------------------------------------------------------
```

```
342 // High priority interrupt vector
343 void IntVecHi (void)
344 {
345   _asm
346     goto ISRHI //jump to interrupt routine
347   _endasm
348 }
349 //---------------------------------------------------------------------------
350
351
352 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
353 #pragma code INTLO = 0x818
354 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
355 //---------------------------------------------------------------------------
356 void IntVecLo (void)
357 {
358   _asm
359     goto ISRLO //jump to interrupt routine
360   _endasm
361 }
362 //---------------------------------------------------------------------------
363
364
365 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
366 #pragma code MAIN
367 //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
368 //---------------------------------------------------------------------------
369 // EnableInterrupts - Configure and then enable active interrupts
370 //
371 //  Currently all supported interrupts are high priority interrupts.
372 //
373 //  CAVEAT EMPTOR: To reduce interrupt latency, none of the floating point
374 //      registers are saved during context save.  So, .. if you replace or
375 //      add interrupt handlers with ones that use these registers you will
376 //      need to change the #pragma directive for ISRHI().
377 //---------------------------------------------------------------------------
378 void EnableInterrupts(void)
379 {
380 // Disable All Interrupts First //
381     INTCONbits.GIEL = 0;
382     INTCONbits.GIEH = 0;
383
384     RCONbits.IPEN = 1;       // Enable Interrupt Priority Mode
385 //  INTCON2 = 0xFF;
386
387 ////////////////////////////
388 // Clear All Interrupt Flags //
389 ////////////////////////////
390     INTCONbits.RBIF = 0;         // RB   - PORT B Change
391     INTCONbits.TMR0IF = 0;       // TMR0
392     INTCONbits.INT0IF = 0;       // INT0
393     INTCON3bits.INT1IF = 0;      // INT1
394     INTCON3bits.INT2IF = 0;      // INT2
395     INTCON3bits.INT3IF = 0;      // INT3
396
397     PIR1 = 0;                    // PSP  - Parallel Slave Port
398                                  // AD   - ADC
399                                  // RC1  - USART #1 Receive
400                                  // TX1  - USART #1 Transmit
401                                  // SSP1 - MSSP #1
402                                  // CCP1 - Capture/Compare/PWM #1
403                                  // TMR2
404                                  // TMR1
405
406     PIR2 = 0;                    // OSCF - Oscillator Fail
407                                  // CM   - Comparator
408                                  // UNUSED
409                                  // EE   - EEPROM/Flash Write
410                                  // BCL1 - MSSP #1 Bus Collision
411                                  // HLVD - High/Low Voltage Detect
412                                  // TMR3
413                                  // CCP2 - Capture/Compare/PWM #2
414
```

```
415     PIR3 = 0;                       // SSP2 – MSSP #2
416                                     // BCL2 – MSSP #2 Bus Collision
417                                     // RC2  – USART #2 Receive
418                                     // TX2  – USART #2 Transmit
419                                     // TMR4
420                                     // CCP5 – Capture/Compare/PWM #5
421                                     // CCP4 – Capture/Compare/PWM #4
422                                     // CCP3 – Capture/Compare/PWM #3
423
424
425 ////////////////////////////////
426 // Configure interrupt priority //
427 //                              //
428 // NOTE: INT0 always HI         //
429 ////////////////////////////////
430     INTCON2bits.TMR0IP = 1;     // TMR0: HI
431     INTCON2bits.INT3IP = 1;     // INT3: HI
432     INTCON2bits.RBIP = 1;       // RB:   HI
433
434     INTCON3bits.INT2IP = 1;     // INT2: HI
435     INTCON3bits.INT1IP = 1;     // INT1: HI
436
437
438     IPR1 = 0xFF;                // PSP:  HI
439                                 // AD:   HI
440                                 // RC1:  HI
441                                 // TX1:  HI
442                                 // SSP1: HI
443                                 // CCP1: HI
444                                 // TMR2: HI
445                                 // TMR1: HI
446
447     IPR2 = 0xFF;                // OSCF: HI
448                                 // CM:   HI
449                                 // UNUSED
450                                 // EE:   HI
451                                 // BCL1: HI
452                                 // HLVD: HI
453                                 // TMR3: HI
454                                 // CCP2: HI
455
456     IPR3 = 0xFF;                // SSP2: HI
457                                 // BCL2: HI
458                                 // RC2:  HI
459                                 // TX2:  HI
460                                 // TMR4: HI
461                                 // CCP5: HI
462                                 // CCP4: HI
463                                 // CCP3: HI
464
465
466 ////////////////////////////
467 // Configure Edge Detect //
468 ////////////////////////////
469     INTCON2bits.INTEDG0 = 1;    // INT0: Rising edge
470     INTCON2bits.INTEDG1 = 1;    // INT1: Rising edge
471     INTCON2bits.INTEDG2 = 1;    // INT2: Rising edge
472     INTCON2bits.INTEDG3 = 1;    // INT3: Rising edge
473
474
475 ////////////////////////////////
476 // Enable Selected Interrupts //
477 ////////////////////////////////
478     INTCONbits.TMR0IE = 1;      // TMR0: Heartbeat: 1ms
479
480                                 // INT0: AB2 – Mtr2 quadrature encoder
481     if(MTR2_MODE1 & analogfbMsk)
482         INTCONbits.INT0IE = 0;
483     else INTCONbits.INT0IE = 1;
484
485                                 // INT1: AB1 – Mtr1 quadrature encoder
486     if(MTR1_MODE1 & analogfbMsk)
487         INTCON3bits.INT1IE = 0;
```

```
488     else INTCON3bits.INT1IE = 1;
489
490     if(MTR2_MODE3 & OcieMsk)     // INT2: I2 - Mtr2 current sense
491         INTCON3bits.INT2IE = 1;
492     else INTCON3bits.INT2IE=0;
493
494     if(MTR1_MODE3 & OcieMsk)     // INT3: I1 - Mtr1 current sense
495         INTCON3bits.INT3IE = 1;
496     else INTCON3bits.INT3IE=0;
497
498     INTCONbits.RBIE = 0;         // RB:   EX2
499
500
501     PIE1bits.PSPIE = 0;          // PSP:  UNUSED
502     PIE1bits.ADIE = 0;           // AD:   UNUSED
503     PIE1bits.RC1IE = 1;          // RC1:  RC1 - Cmd Interface
504     PIE1bits.TX1IE = 1;          // TX1:  TX1 - Cmd Interface
505     PIE1bits.SSP1IE = 0;         // SSP1: UNUSED
506     PIE1bits.CCP1IE = 1;         // CCP1: EX0 Pulse capture (R/C Mtr1)
507     PIE1bits.TMR2IE = 1;         // TMR2: ADC State Machine
508     PIE1bits.TMR1IE = 1;         // TMR1: RTC; Timed delays
509
510     PIE2bits.OSCFIE = 0;         // OSCF: UNUSED
511     PIE2bits.CMIE = 0;           // CM:   UNUSED
512                                  // UNUSED
513     PIE2bits.EEIE = 0;           // EE:   UNUSED
514     PIE2bits.BCL1IE = 0;         // BCL1: UNUSED
515     PIE2bits.HLVDIE = 0;         // HLVD: UNUSED
516     PIE2bits.TMR3IE = 0;         // TMR3: Capture/Compare - All CCP Modules
517     PIE2bits.CCP2IE = 1;         // CCP2: EX1 Pulse capture (R/C Mtr2)
518
519     PIE3bits.SSP2IE = 1;         // SSP2: Secondary I2C bus.
520     PIE3bits.BCL2IE = 0;         // BCL2: UNUSED
521     PIE3bits.RC2IE = 0;          // RC2:  UNUSED
522     PIE3bits.TX2IE = 0;          // TX2:  UNUSED
523     PIE3bits.TMR4IE = 0;         // TMR4: PWM - All CCP Modules
524     PIE3bits.CCP5IE = 0;         // CCP5: PWM1 - Mtr1 speed control
525     PIE3bits.CCP4IE = 0;         // CCP4: PWM2 - Mtr2 speed control
526     PIE3bits.CCP3IE = 0;         // CCP3: EX3
527
528
529     TXSTA1bits.TXEN = 1;         // USART1: Tx enable (sets TX1IF)
530     INTCON2bits.RBPU = 1;        // RB pull-ups disabled.
531
532 ////////////////////////////
533 // Enable Global Interrupts //
534 ////////////////////////////
535     INTCONbits.GIEL = 0;     // Disable Low priority interrupts
536     INTCONbits.GIEH = 1;     // Enable high priority interrupts
537 }
538 //-----------------------------------------------------------------------------
539
540
541 ////////////////////////////////////////////////////////////////////////////
542 // _user_putc - Single char output in an application defined manner
543 //
544 //      Printf (et. al.) to an "output stream".  MCC18 defines 2 such streams:
545 //
546 //            _H_USER -  Output via user-defined function _user_putc.
547 //            _H_USART - Output via library output function _usart_putc.
548 //
549 //      By defining _user_putc, Printf funnels all its output thru this fcn.
550 //      The use of TxChr() in this way allows all USART1 output (including that
551 //      from the assembler) to be interrupt driven using a common transmit
552 //      buffer (Tx1_Buff[]).
553 //-----------------------------------------------------------------------------
554 int _user_putc (char c)
555 {
556     return ( (int) TxChr(c) );
557 }
558 //-----------------------------------------------------------------------------
559 void     Greeting(void)
560 {
```

```
561     if(SCFG == TEcfg)
562     { // If Terminal Emulator Interface
563         printf("\r\n");
564         printf("Dalf-1%c\r\n", BOARD_ID);                          // Hardware ID
565         printf("18F6722 Rev:%02X\r\n", (PIC_DEVID1 & 0x1F));       // Micro ID
566         printf("Software Ver:%2u.%02u\r\n",MAJOR_ID, MINOR_ID);    // Software ID
567         printf("User: %05u\r\n",USERID);                           // User ID
568     }
569 }
570 //-------------------------------------------------------------------------
571 void    SerialCmdDispatch(void)
572 { // If a serial command has been received, try to execute it.
573     BYTE err;
574
575     if(serialcmd)
576     { // If cmd received by one of serial interfaces
577         //-------------------------------------
578         if( CmdSource == API_SrcMsk )
579         { // If API
580             err = ApiCmdDispatch();
581         }
582         //-------------------------------------
583         else if (CmdSource == TE_SrcMsk)
584         { // Terminal Emulator Interface
585             err = TeCmdDispatchExt();
586             if (err)
587             {
588                 if (err==eParseErr)     printf("Parse Err\r\nEnter 'H' for help\r\n");
589                 if (err==eNumArgsErr)   printf("#Args Err\r\nEnter 'H' for help\r\n");
590                 if (err==eParmErr)      printf("Parm Err\r\nEnter 'H' for help\r\n");
591                 if (err==eModeErr)      printf("Mode Err\r\n");
592                 if (err==eDisable)      printf("Disabled\r\n");
593             }
594         }
595         //-------------------------------------
596         else if (CmdSource == I2C2_SrcMsk)  err = I2C2CmdDispatchExt();
597         //-------------------------------------
598
599         serialcmd=FALSE;          // reset flag
600         if(!err) Cmd_Ticks = CMDTIME;    // Reset Cmd Interface Timeout ticker
601     }
602 }
603 //-------------------------------------------------------------------------
604 void    PotOrRc(BYTE mtr)
605 { // Does nothing unless RC or POT mode and service is due (TIMESVC).
606     BYTE mode2, MtrMode;
607
608
609     if(mtr==1)  mode2=MTR1_MODE2;    else mode2=MTR2_MODE2;
610     if((mode2 & ManualMsk)==0) MtrMode = RS232; // If not RC or POT
611     else
612     {
613         if(mode2 & PotcMsk)          MtrMode = POT_CENTER_OFF;
614         if(mode2 & PotfMsk)          MtrMode = POT_WITH_SWITCH;
615         if(mode2 & RcNrmMsk)         MtrMode = RADIO_NORMAL;
616         if(mode2 & RcMixMsk)         MtrMode = RADIO_MIX;
617         if(mode2 & RcServoMsk)       MtrMode = RADIO_SERVO;
618         if(mode2 & PotServoMsk)      MtrMode = POT_SERVO;
619         if(mode2 & PotMixMsk)        MtrMode = POT_MIX;
620     }
621
622     switch(MtrMode)
623     {
624     //-----------------------------------------------------
625     case POT_CENTER_OFF:                                // Open Loop
626         if(TIMESVC & PspMsk)     PotCcmd(mtr);
627         break;
628     //-----------------------------------------------------
629     case POT_WITH_SWITCH:                               // Open Loop
630         if(TIMESVC & PspMsk)     PotFcmd(mtr);
631         break;
632     //-----------------------------------------------------
633     case POT_SERVO:                                     // Closed Loop
```

```
634             if(TIMESVC & PspMsk)      PotServoCmd(mtr);
635             break;
636         //-------------------------------------------------------
637         case RADIO_NORMAL:                                  // Open Loop
638             if(TIMESVC & RcspMsk)     RcCmd(mtr);
639             break;
640         //-------------------------------------------------------
641         case RADIO_MIX:                                     // Open Loop
642             if((TIMESVC & RcspMsk) && (mtr==1)) RcMixCmd();
643             break;
644         //-------------------------------------------------------
645         case RADIO_SERVO:                                   // Closed Loop
646             if(TIMESVC & RcspMsk)     RcServoCmd(mtr);
647             break;
648         //-------------------------------------------------------
649         case POT_MIX:                                       // Open Loop
650             if((TIMESVC & PspMsk) && (mtr==1))   PotMixCmd();
651             break;
652         //-------------------------------------------------------
653         default:
654             break;
655         }
656 }
657 //-----------------------------------------------------------------------------
658 void     PotCcmd(BYTE mtr)
659 {
660     BYTE dir,spd,adc;
661
662     CmdSource = POT_SrcMsk;
663     if(SWITCH0)
664     {
665         if(mtr==1) adc=ADC0[0]; else adc=ADC0[1];
666         spd = PotCtoPWM(adc);
667         if((spd) && !(adc&0x80)) dir=REVERSE;    // If 0<spd  AND adc<0x080
668         else dir=FORWARD;                        // else
669         MoveMtrOpenLoop(mtr, dir, spd, AMINP);
670     }
671     else     SoftStop(mtr);       // Schedule Motor Stop
672 }
673 //-----------------------------------------------------------------------------
674 void     PotFcmd(BYTE mtr)
675 {
676     BYTE dir,spd,adc;
677
678     CmdSource = POT_SrcMsk;
679     if(SWITCH0)
680     {
681         if(mtr==1) adc=ADC0[0]; else adc=ADC0[1];
682         spd = PotFtoPWM(adc);
683         if( spd && !SWITCH1) dir=REVERSE;    // if 0<spd AND sw1=0
684         else dir=FORWARD;                    // else
685         MoveMtrOpenLoop(mtr, dir, spd, AMINP);
686     }
687     else     SoftStop(mtr);       // Schedule Motor Stop
688 }
689 //-----------------------------------------------------------------------------
690 void PotServoCmd(BYTE mtr)
691 {
692     ////////////////////////////////////////////////////////////////////////
693     // If enabled by switch (PORTD.0), ...                                //
694     //   Map ADC reading "pot" in [0..255] to y position in [MIN..MAX] and use  //
695     //   PID to move to position y.  Initial move uses Trajectory Generator for  //
696     //   smooth startup.  After that, repeated PID w/o err sum term.      //
697     ////////////////////////////////////////////////////////////////////////
698     BYTE pot,state,flags;
699     short long y;
700     WORD min,max,vmid,acc;
701
702     if(mtr==1)
703     { // If Motor1
704         min=MIN1; max=MAX1; vmid=VMID1; acc=ACC1;
705         flags=Mtr1_Flags1; pot=ADC0[0];
706     }
```

```
    else
    { // Else Motor2
        min=MIN2; max=MAX2; vmid=VMID2; acc=ACC2;
        flags=Mtr2_Flags1; pot=ADC0[1];
    }

    if(SWITCH0)
    { // If Enabled by external switch
        y = max - min;
        y = (y*pot)/255 + min;          // target position

        state=0;
        if( flags & pida_Msk ) state+=1;    // If PID active
        if( flags & tga_Msk ) state+=2;     // If TGA active
        switch(state)
        {
            case 0: // tga OFF, pid OFF
                CmdSource = POT_SrcMsk;
                MoveMtrClosedLoop(mtr, y, vmid, acc);
                break;

            case 1: // tga OFF, pid ON
                if(mtr==1)
                { // If Motor1
                    x1pos=y;                    // New pid target
                    Mtr1_Flags1 |= sumhoa_Msk;  // Don't accumulate PID Err Sums
                }
                else
                { // Else Motor2
                    x2pos=y;
                    Mtr2_Flags1 |= sumhoa_Msk;
                }
                break;

            case 2: // tga ON, pid OFF
                break;                          // Just exit

            case 3: // tga ON, pid ON
                break;                          // Exit. Let 1st target be reached.
            default:
                break;
        } // state
    } // SWITCH0
    else
    {
        CmdSource = POT_SrcMsk;
        SoftStop(mtr);      // Schedule Motor Stop
    }
}
//-------------------------------------------------------------------------
void    RcCmd(BYTE mtr)
{
    ////////////////////////////////////////////////////////////////
    //  Maps measured pulse width pw to open loop motor control   //
    //  parameters SPD and DIR which are used to issue the motor  //
    //  open loop command.  AMINP is used as the slew rate.  The  //
    //  mapping is the normal one (sometimes called "tank mode").  //
    //                                                            //
    //  RCm is the precomputed mapping slope scaled by 1024.  See  //
    //  RCtoPWM() for details.                                    //
    ////////////////////////////////////////////////////////////////
    BYTE dir, spd;
    WORD rcmin,rcmax,slope,pw;
    int pwm;

    ////////////////////////////////////////////////
    //  First a bit of setup and signal loss detection  //
    ////////////////////////////////////////////////
    if(mtr==1)
    { // If Motor1
        rcmin=RC1MIN; rcmax=RC1MAX; slope=RC1m; pw=RC[0];
        if((NewPulse & NewRC1msk) == 0)     // If Signal Loss, ..
        {
```

```
780             // Schedule Mtr1 Stop
781             CmdSource = RC_SrcMsk;
782             SoftStop(0x01);
783             LedErr |= SL1msk;        // Record in LedErr
784             return;                  // .. and exit.
785         }
786         else
787         {
788             NewPulse &= ~NewRC1msk; // Reset NewPulse.RC1
789             LedErr &= ~SL1msk;       // Reset also in LedErr
790         }
791     }
792     else
793     { // Else Motor2
794         rcmin=RC2MIN; rcmax=RC2MAX; slope=RC2m; pw=RC[1];
795         if((NewPulse & NewRC2msk) == 0)      // If Signal Loss, ..
796         {
797             // Schedule Mtr2 Stop
798             CmdSource = RC_SrcMsk;
799             SoftStop(0x02);
800             LedErr |= SL2msk;        // Record in LedErr
801             return;                  // .. and exit.
802         }
803         else
804         {
805             NewPulse &= ~NewRC2msk; // Reset NewPulse
806             LedErr &= ~SL2msk;       // Reset also in LedErr
807         }
808     }
809
810
811     // Get pwm value [-100, 100] corresponding to pulse width.
812     pwm = RCtoPWM(pw, rcmin, rcmax, slope);
813
814     // Convert pwm value into DIR and SPD control parameters.
815     if (pwm < 0) {dir = REVERSE; spd = -pwm;}
816     else {dir = FORWARD; spd = pwm;}
817
818     // Issue Mtr Move Cmd (Open Loop)
819     CmdSource = RC_SrcMsk;
820     MoveMtrOpenLoop(mtr, dir, spd, AMINP);
821 }
822 //-----------------------------------------------------------------------------
823 void    RcMixCmd(void)
824 {
825     ////////////////////////////////////////////////////////////////
826     //  Maps measured pulse widths RC[0], RC[1] to corresponding   //
827     //  global pwm values pwm1 and pwm2.  These are then "mixed"   //
828     //  to produce open loop motor command parameters for both     //
829     //  motors.  AMINP is used as the slew rate.                   //
830     //                                                             //
831     //  Parms: RC1MIN, RC1MAX, and RC1m affect the pwm1 mapping.   //
832     //  Parms: RC2MIN, RC2MAX, and RC2m affect the pwm2 mapping.   //
833     //                                                             //
834     //  Just exits if either motor not in RCMIX mode.              //
835     ////////////////////////////////////////////////////////////////
836     BYTE dir1, spd1, dir2, spd2;
837     int pwm;
838
839     ////////////////////////////////////////////////
840     //  First a bit of setup and signal loss detection  //
841     ////////////////////////////////////////////////
842     if(  ((NewPulse & NewRC1msk)==0) || ((NewPulse & NewRC2msk) == 0) )
843     { // If Signal Loss, ...
844         if((NewPulse & NewRC1msk)==0)   LedErr |=SL1msk;    // Record in LedErr
845         if((NewPulse & NewRC2msk)==0)   LedErr |=SL2msk;
846         CmdSource = RC_SrcMsk;
847         SoftStop(0x01);          // Schedule stop of Mtr1
848         SoftStop(0x02);          // Schedule stop of Mtr2
849         return;                  // .. and exit.
850     }
851     else
852     {
```

```
853             NewPulse &= ~(NewRC1msk + NewRC2msk);    // Reset NewPulse.(RC1+RC2)
854             LedErr &= ~(SL1msk + SL2msk);            // Reset also in LedErr
855         }
856         if(!(MTR1_MODE2 & RcMixMsk))     return;     // Just exit if not RCMIX.
857         if(!(MTR2_MODE2 & RcMixMsk))     return;     // Just exit if not RCMIX.
858
859
860         // Get pwm values [-100, 100] corresponding to pulse widths.
861         pwm1 = RCtoPWM(RC[0], RC1MIN, RC1MAX, RC1m);
862         pwm2 = RCtoPWM(RC[1], RC2MIN, RC2MAX, RC2m);
863
864         // Convert pwm values into DIR and SPD control parameters.
865         pwm = pwm1 - pwm2;
866         if (pwm < -100) pwm = -100;
867         if (pwm > 100) pwm = 100;
868         if (pwm < 0) {dir1 = REVERSE; spd1 = -pwm;}
869         else {dir1 = FORWARD; spd1 = pwm;}
870
871         pwm = pwm1 + pwm2;
872         if (pwm < -100) pwm = -100;
873         if (pwm > 100) pwm = 100;
874         if (pwm < 0) {dir2 = REVERSE; spd2 = -pwm;}
875         else {dir2 = FORWARD; spd2 = pwm;}
876
877         // Issue Mtr1 Move Cmd (Open Loop)
878         CmdSource = RC_SrcMsk;
879         MoveMtrOpenLoop(0x01, dir1, spd1, AMINP);
880
881         // Issue Mtr2 Move Cmd (Open Loop)
882         CmdSource = RC_SrcMsk;
883         MoveMtrOpenLoop(0x02, dir2, spd2, AMINP);
884 }
885 //-------------------------------------------------------------------------
886 void PotMixCmd(void)
887 {
888     ////////////////////////////////////////////////////////////////////
889     //  Maps measured adc values ADC0[0], ADC1[1] to corresponding    //
890     //  pwm values pwm1 and pwm2.  These values are then "mixed"      //
891     //  to produce open loop motor command parameters for both        //
892     //  motors.  AMINP is used as the slew rate.                      //
893     //                                                                //
894     //  NOTES:                                                        //
895     //  1) Just exits if either motor not in POTMIX mode.             //
896     //  2) Configure Mtr1 as right motor as seen from drivers seat.   //
897     ////////////////////////////////////////////////////////////////////
898     BYTE dir1, spd1, dir2, spd2, adc;
899     int pwm;
900
901     if(!(MTR1_MODE2 & PotMixMsk))    return;     // Just exit if not POTMIX.
902     if(!(MTR2_MODE2 & PotMixMsk))    return;     // Just exit if not POTMIX.
903
904     ////////////////////////////////////////////////////////////
905     //  Capture Pot readings and convert to [-100%, 100%] pwm //
906     ////////////////////////////////////////////////////////////
907     adc = ADC0[0];
908     pwm1 = PotCtoPWM(adc);                  // [0,100%]
909     if(adc<0x80) pwm1 = -pwm1;              // pwm1: [-100%, 100%]
910     adc = ADC0[1];
911     pwm2 = PotCtoPWM(adc);                  // [0,100%]
912     if(adc<0x80) pwm2 = -pwm2;              // pwm2: [-100%, 100%]
913
914     ////////////////////////////////////////////////////////////
915     // Convert pwm values into DIR and SPD control parameters. //
916     ////////////////////////////////////////////////////////////
917     pwm = pwm1 - pwm2;
918     if (pwm < -100) pwm = -100;
919     if (pwm > 100) pwm = 100;
920     if (pwm < 0) {dir1 = REVERSE; spd1 = -pwm;}
921     else {dir1 = FORWARD; spd1 = pwm;}
922
923     pwm = pwm1 + pwm2;
924     if (pwm < -100) pwm = -100;
925     if (pwm > 100) pwm = 100;
```

```
926     if (pwm < 0) {dir2 = REVERSE; spd2 = -pwm;}
927     else {dir2 = FORWARD; spd2 = pwm;}
928
929     if(SWITCH0)
930     {    // Safety check: On/Off switch
931          // Issue Mtr1 Move Cmd (Open Loop)
932          CmdSource = POT_SrcMsk;
933          MoveMtrOpenLoop(0x01, dir1, spd1, AMINP);
934
935          // Issue Mtr2 Move Cmd (Open Loop)
936          CmdSource = POT_SrcMsk;
937          MoveMtrOpenLoop(0x02, dir2, spd2, AMINP);
938     }
939     else
940     {
941          CmdSource = POT_SrcMsk;
942          SoftStop(1);          // Schedule Motor Stop
943          CmdSource = POT_SrcMsk;
944          SoftStop(2);          // Schedule Motor Stop
945     }
946 }
947 //---------------------------------------------------------------------------
948 void RcServoCmd(BYTE mtr)
949 {
950     ////////////////////////////////////////////////////////////////////////
951     //  Maps measured R/C pulse width pw= in [RCMIN..RCMAX] to y in       //
952     //  in [MIN..MAX] and use PID to move motor to position y.            //
953     //                                                                    //
954     //  Initial move uses Trajectory Generator for smooth startup.        //
955     //  After that, repeated PID w/o err sum term.                        //
956     //                                                                    //
957     //             slope = 100*(MAX-MIN)/RCMAX-RCMIN)                     //
958     //                                                                    //
959     //  Precomputed mapping slope scaled by 100.  Avoids some unnecessary //
960     //  runtime computations.                                             //
961     ////////////////////////////////////////////////////////////////////////
962     BYTE state, flags;
963     WORD pw,min,rcmax,rcmin,vmid,acc,slope;
964     short long y;
965
966     ////////////////////////////////////////////////////
967     //  First a bit of setup and signal loss detection //
968     ////////////////////////////////////////////////////
969     if(mtr==1)
970     { // If Motor1
971          min=MIN1; rcmax=RC1MAX; rcmin=RC1MIN; vmid=VMID1; acc=ACC1;
972          flags=Mtr1_Flags1; slope=RC1M;  pw=RC[0];
973
974          if((NewPulse & NewRC1msk) == 0)      // If Signal Loss, ..
975          {
976               CmdSource = RC_SrcMsk;
977               SoftStop(0x01);          // Schedule stop of Mtr
978               LedErr |= SL1msk;        // Record in LedErr
979               return;                  // ... and exit
980          }
981          else
982          {
983               NewPulse &= ~NewRC1msk; // Reset NewPulse.RC1
984               LedErr &= ~SL1msk;       // Reset also in LedErr
985          }
986     }
987     else
988     { // Else Motor2
989          min=MIN2; rcmax=RC2MAX; rcmin=RC2MIN; vmid=VMID2; acc=ACC2;
990          flags=Mtr2_Flags1;  slope=RC2M; pw=RC[1];
991
992          if((NewPulse & NewRC2msk)==0)        // If Signal Loss, ..
993          {
994               CmdSource = RC_SrcMsk;
995               SoftStop(0x02);          // Schedule stop of Mtr
996               LedErr |= SL2msk;        // Record in LedErr
997               return;                  // ... and exit
998          }
```

```
999         else
1000        {
1001            NewPulse &= ~NewRC2msk; // Reset NewPulse.RC2
1002            LedErr &= ~SL2msk;      // Reset also in LedErr
1003        }
1004    }
1005
1006
1007    // Clip pulse width to [RCMIN..RCMAX] range.
1008    if(pw < rcmin) pw=rcmin;
1009    if(pw > rcmax) pw=rcmax;
1010    y = (slope*(pw - rcmin)/100)+min;       // target position
1011
1012    // Mapping and dispatch based on PID and TG
1013    state=0;
1014    if( flags & pida_Msk )  state+=1;       // if PID active
1015    if( flags & tga_Msk )   state+=2;       // if TG active
1016    switch(state)
1017    {
1018        case 0: // tga OFF, pid OFF
1019            CmdSource = RC_SrcMsk;
1020            MoveMtrClosedLoop(mtr, y, vmid, acc);
1021            break;
1022
1023        case 1: // tga OFF, pid ON
1024            if(mtr==1)
1025            { // If Motor1
1026                x1pos=y;                    // New pid target
1027                Mtr1_Flags1 |= sumhoa_Msk;  // Don't accumulate PID Err Sums
1028            }
1029            else
1030            { // Else Motor2
1031                x2pos=y;
1032                Mtr2_Flags1 |= sumhoa_Msk;
1033            }
1034            break;
1035
1036        case 2: // tga ON, pid OFF
1037            break;                      // Just exit
1038
1039        case 3: // tga ON, pid ON
1040            break;                      // Exit. Let 1st target be reached.
1041        default:
1042            break;
1043    } // state
1044 }
1045 //---------------------------------------------------------------------------
1046 void CheckCmdTimeout(void)
1047 {
1048     ////////////////////////////////////////////////////////
1049     //  Check for timeout on the Serial CMD Interfaces    //
1050     //                                                    //
1051     //  (Don't stop motor if in one of the R/C or POT modes)  //
1052     //                                                    //
1053     //  If timeout: Stop motors not under POT or R/C controls.  //
1054     //              Reset timeout counter.                //
1055     //                                                    //
1056     //        Else: Decrement timeout counter.            //
1057     ////////////////////////////////////////////////////////
1058     if(SYSMODE & CmdToMsk)  // If timeout feature enabled
1059     {
1060         if(Cmd_Ticks == 0)
1061         {
1062             if(!(MTR1_MODE2 & ManualMsk)) // if not R/C or POT controlled
1063             {
1064                 CmdSource = TE_SrcMsk;
1065                 SoftStop(1);
1066             }
1067             if(!(MTR2_MODE2 & ManualMsk)) // if not R/C or POT controlled
1068             {
1069                 CmdSource = TE_SrcMsk;
1070                 SoftStop(2);
1071             }
```

```
1072                Cmd_Ticks = CMDTIME;
1073            }
1074            else Cmd_Ticks--;
1075        }
1076    }
1077    //-------------------------------------------------------------------------
1078
1079
1080
1081
1082
1083
1084    //-------------------------------------------------------------------------
1085    BYTE    PotCtoPWM(BYTE adc)
1086    {
1087        WORD y=212;
1088        ////////////////////////////////////////////////////////////////////
1089        // Maps pot ADC reading into 0-100% speed assuming center position  //
1090        // adc = 0x7F should be mapped to 0% with speed increasing linearly //
1091        // to either side of zero.  With a deadband around center reading   //
1092        // of +/- 0x06, the slope is                                        //
1093        //                                                                  //
1094        //                  Slope = 100/(121) ~ 212/256                     //
1095        ////////////////////////////////////////////////////////////////////
1096        if(adc>0x85) y=(y*(adc-0x085))>>8;
1097        else if(adc<0x7A) y=(y*(0x7A-adc))>>8;
1098        else    y=0;
1099        if(y>100) y=100;        // clip to full range.
1100        return  (BYTE) y;
1101    }
1102    //-------------------------------------------------------------------------
1103    BYTE    PotFtoPWM(BYTE adc)
1104    {
1105        WORD y=103;
1106        ////////////////////////////////////////////////////////////////////
1107        // Maps pot ADC reading into 0-100% speed assuming full range on the //
1108        // pot reading [0..255] maps to [0..100%] speed setting.  A separate //
1109        // switch controls direction.                                       //
1110        //                                                                  //
1111        // adc is mapped linearly to full pwm range.                        //
1112        // Slope = 100/249 ~ 103/256                                        //
1113        //                                                                  //
1114        // Also provides deadband around 0x00 reading of + 0x06             //
1115        ////////////////////////////////////////////////////////////////////
1116        if(adc>0x06) y=(y*(adc-0x06))>>8;
1117        else y=0;
1118        if(y>100) y=100;        // clip to full range.
1119        return  (BYTE) y;
1120    }
1121    //-------------------------------------------------------------------------
1122    int RCtoPWM(WORD p, WORD rcmin, WORD rcmax, WORD rcm)
1123    {
1124        ////////////////////////////////////////////////////////////////////
1125        //      p = measured pulse width (microseconds)                   //
1126        //      rcmin = minimum pulse width (microseconds)                //
1127        //      rcmax = maximum pulse width (microseconds)                //
1128        //      rcm = scaled mapping slope of PWM% vs pulse width         //
1129        //                                                                //
1130        //        Returns signed integer in range [-100, 100].            //
1131        //                                                                //
1132        //  rcm is the precomputed mapping slope scaled by 1024.          //
1133        //  Usage avoids some unnecessary runtime computations.           //
1134        //                                                                //
1135        //          rcm = [200/(rcmax - rcmin - d)]*1024                  //
1136        //                                                                //
1137        //            d = RCD (R/C Deadband parameter)                    //
1138        ////////////////////////////////////////////////////////////////////
1139        long z1;
1140        WORD cx,dx;
1141        int rtnval;
1142
1143        if (p<rcmin) return(-100);
1144        if (p>rcmax) return(100);
```

```
1145
1146      cx = (rcmin+rcmax)>>1;       // cx = Center of expected pulse width range
1147      dx = RCD>>1;                 // dx = deadband/2
1148      if ( p<(cx-dx) )
1149      { // return( [rcm/1024]*(p-rcmin) - 100 )
1150          z1 = p-rcmin;
1151          z1 = z1*(long)rcm;
1152          z1 = z1>>10;                     // unscale
1153          rtnval = (int)z1 - 100;
1154          if (rtnval<-100) return(-100);
1155          return(rtnval);
1156      }
1157      else if ( p>(cx+dx) )
1158      { // return( [rcm/1024]*(p-(cx+dx)) )
1159          z1 = p - (cx+dx);
1160          z1 = z1*(long)rcm;
1161          z1 = z1>>10;                     // unscale
1162          rtnval = (int)z1;
1163          if (rtnval>100) return(100);
1164          return(rtnval);
1165      }
1166      else return(0);
1167  }
1168  //-------------------------------------------------------------------------
1169  WORD AdcConvert(WORD Adc)
1170  ////////////////////////////////////////////////////////////////////
1171  //  Converts ADC reading [0..255] to [0..5000] millivolts.        //
1172  //      V(mV) = (5000/255)*AdcReading.                            //
1173  //      Note: 5000/255 = 19.607843.. = 19 + 155/256 + 0.002374..  //
1174  ////////////////////////////////////////////////////////////////////
1175  {
1176      return (19*Adc + ((155*Adc)>>8));
1177  }
1178  //-------------------------------------------------------------------------
1179  WORD PulseConvert(WORD Pulse)
1180  {
1181  ////////////////////////////////////////////////////////////////////
1182  //  Converts Pulse Width to units of 1.00 uSec                    //
1183  //      (Argument is in units of 0.80 uSec)                       //
1184  //                                                                //
1185  //  Pulse_uSec = 0.80 * PulseWidth.                               //
1186  //             = (204/256 + 204/256^2 + 0.000012..) * PulseWidth  //
1187  //                                                                //
1188  ////////////////////////////////////////////////////////////////////
1189      unsigned short long p=Pulse;
1190      p=p*204;
1191      return( (WORD)((p>>8) + (p>>16)) );
1192  }
1193  //-------------------------------------------------------------------------
1194  WORD AdcToVbatt(WORD v6)
1195  {
1196      ////////////////////////////////////////////////////////////////
1197      //          Compute actual VBATT voltage (millivolts)         //
1198      //                                                            //
1199      // VBATT volt divider constant:  r=R3/(R3+R4)=0.130435        //
1200      // is used to translate voltage V6 measured by ADC on AN6     //
1201      // into actual VBATT voltage.                                 //
1202      //                                                            //
1203      //                VBATT = (1/r) * V6                          //
1204      //                                                            //
1205      //   Note: 1/r ~ 7.666.. = 7 + 170/256 + 0.002604..           //
1206      //                                                            //
1207      //   To deal with resistor variability, the scaled calibration //
1208      //   constant VBCAL is stored in the Parameter Block.         //
1209      //                                                            //
1210      //           VBCAL = 256 * (1/r)  [default: 0x07AA]           //
1211      //                                                            //
1212      //   Note: VBCAL adjustment in the Parameter Block enables fine //
1213      //   tuning in this computation of the VBATT voltage.         //
1214      ////////////////////////////////////////////////////////////////
1215      unsigned short long temp;
1216      temp = v6;                           // V6 (mv)
1217      return( (WORD)((VBCAL*temp)>>8) );  // Multiply and unscale
```

```
1218 }
1219 //---------------------------------------------------------------------------
1220
1221
1222
1223
1224
1225 //---------------------------------------------------------------------------
1226 void SoftStop(BYTE mtr)
1227 {
1228     ////////////////////////////////////////////////////////
1229     // Schedules ramped stop of the motor.                //
1230     //                                                    //
1231     // This function uses Cmd_X to stop the motor.  Unlike //
1232     // Cmd_O, it will not disable R/C and POT mode operations //
1233     // provided that CmdSource is defined properly.       //
1234     ////////////////////////////////////////////////////////
1235     #ifdef DALF_ROVIM_T2D
1236     if(mtr==3)
1237     {
1238         //Stop traction motor and set it on hill hold
1239         CMD = 'G';
1240         ARG[0] = ROVIM_T2D_SET_MOVEMENT_CMD_CODE;
1241         ARGN = 1;
1242         TeCmdDispatchExt();
1243         return;
1244     }
1245     #endif  //DALF_ROVIM_T2D
1246     CMD = 'X';
1247     ARG[0] = mtr;
1248     ARG[1] = FORWARD;
1249     ARG[2] = SPEEDZERO;
1250     ARG[3] = AMINP;
1251     ARGN = 0x04;
1252     TeCmdDispatchExt();
1253 }
1254 //---------------------------------------------------------------------------
1255 void MoveMtrOpenLoop(BYTE mtr, BYTE dir, BYTE spd, BYTE slew)
1256 {
1257     CMD = 'X';
1258     ARG[0] = mtr;
1259     ARG[1] = dir;
1260     ARG[2] = spd;
1261     ARG[3] = slew;
1262     ARGN = 0x04;
1263     TeCmdDispatchExt();
1264 }
1265 //---------------------------------------------------------------------------
1266 void MoveMtrClosedLoop(BYTE mtr, short long tgt, WORD v, WORD a)
1267 {
1268     CMD = 'Y';
1269     ARG[0] = mtr;
1270     ARG[1] = (tgt & 0xFF0000)>>16;
1271     ARG[2] = (tgt & 0xFF00)>>8;
1272     ARG[3] = tgt & 0xFF;
1273     ARG[4] = (BYTE) (v>>8);
1274     ARG[5] = (BYTE) (v & 0xFF);
1275     ARG[6] = (BYTE) (a>>8);
1276     ARG[7] = (BYTE) (a & 0xFF);
1277     ARGN = 0x08;
1278     TeCmdDispatchExt();
1279 }
1280 //---------------------------------------------------------------------------
1281
1282 void ReturnData(void)
1283 {   // Return data and/or status to command initiator.
1284     if (DispSvc)
1285     { // If anything to do ...
1286     ////////////////////////////////////////////////////////////////////////
1287     //                      -- CAUTION --                                 //
1288     //   DispCHR() is used exclusively to send a single byte (ACKNOWLEDGE)  //
1289     //   back to the command initiator (Serial Command Monitor). The byte   //
1290     //   is either chr_OK (Success) or the ErrCode (Failure) value which    //
```

```
1291      //   serves to identify the problem.                                  //
1292      //                                                                    //
1293      //   In order for commands that return data to respond in the correct //
1294      //   sequence (first ACKNOWLEDGE then packetized data), DispCHR must  //
1295      //   remain first in this list of routines.                           //
1296      ////////////////////////////////////////////////////////////////////////
1297          if (DispSvc & CHRreqMsk)          DispCHR();
1298          if (DispSvc & EreqMsk)            DispE();
1299          if (DispSvc & VreqMsk)            DispV();
1300          if (DispSvc & ADCreqMsk)          DispADC();
1301          if (DispSvc & RCreqMsk)           DispRC();
1302          if (DispSvc & HMSreqMsk)          DispHMS();
1303          if (DispSvc & MEMBYTEreqMsk)      DispMEM();
1304          if (DispSvc & IOBYTEreqMsk)       DispIO();
1305
1306          if (DispSvc & STATreqMsk)         DispSTAT();
1307          if (DispSvc & PIDreqMsk)          DispPID();
1308          if (DispSvc & RAMBLKreqMsk)       DispRAMBLOCK();
1309          if (DispSvc & EXTEEBLKreqMsk)     DispExtEEBLOCK();
1310          if (DispSvc & INTEEBLKreqMsk)     DispIntEEBLOCK();
1311          if (DispSvc & RESETreqMsk)        ResetPIC();
1312      } // If anything to do ...
1313  }
1314  //---------------------------------------------------------------------------
1315  void DispCHR(void)
1316  { // Conditionally Transmit xCHR during serial {API, I2C2} cmd processing.
1317      DispSvc &= ~CHRreqMsk;        // Clear request flag
1318
1319      // If API, always transmit xCHR
1320      if(CmdSource == API_SrcMsk) printf("%c",xCHR);
1321
1322      // If I2C2, transmit xCHR only if (Error) OR (NoError and no data).
1323      else if(CmdSource == I2C2_SrcMsk)
1324      {
1325          if( (xCHR != chr_OK) || (ReturnN == 0) )
1326          {
1327              Pkt[0]=xCHR;              // In case AutoStart needed.
1328              WriteSSP2BUF(Pkt[0]);     // Start Transmission
1329          }
1330      }
1331  }
1332  //---------------------------------------------------------------------------
1333  void DispE(void)
1334  {
1335      short long  E1, E2, E;
1336      ////////////////////////////////////////////////////////////////////
1337      // Remark: The 24-bit, 2's complement encoder counter is updated  //
1338      // within ISR's.  The ISR's are briefly disabled here to avoid    //
1339      // potential glitch in captured values of encoder counters.       //
1340      ////////////////////////////////////////////////////////////////////
1341      DispSvc &= ~EreqMsk;          // Clear request flag
1342
1343      INTCONbits.GIEH = 0;      // Disable high priority interrupts
1344      E1=encode1;
1345      E2=encode2;
1346      INTCONbits.GIEH = 1;      // Enable high priority interrupts
1347
1348      //-----------------------------
1349      if(CmdSource == API_SrcMsk)
1350      { // API Mode
1351          if(DataReq==0xFF)
1352          { // Send Both Motor Positions; Mtr1 first.
1353              PktLen=12;
1354              Pkt[0]=chr_STX;
1355              Pkt[1]=HOST_NID;
1356              Pkt[2]='E';
1357              Pkt[3]=0x06;
1358              Pkt[4]=(E1 & 0xFF);
1359              Pkt[5]=((E1>>8) & 0xFF);
1360              Pkt[6]=((E1>>16) & 0xFF);
1361              Pkt[7]=(E2 & 0xFF);
1362              Pkt[8]=((E2>>8) & 0xFF);
1363              Pkt[9]=((E2>>16) & 0xFF);
```

```
1364                Pkt[11]=chr_ETX;
1365                SendApiPkt();
1366            }
1367            else
1368            { // Single Motor request
1369                if(DataReq==0x01) E=E1; else E=E2;
1370                PktLen=9;
1371                Pkt[0]=chr_STX;
1372                Pkt[1]=HOST_NID;
1373                Pkt[2]='E';
1374                Pkt[3]=0x03;
1375                Pkt[4]=(E & 0xFF);
1376                Pkt[5]=((E>>8) & 0xFF);
1377                Pkt[6]=((E>>16) & 0xFF);
1378                Pkt[8]=chr_ETX;
1379                SendApiPkt();
1380            }
1381        } // If API
1382        //-------------------------------
1383        else if(CmdSource == TE_SrcMsk)
1384        { // TE Mode
1385            if((DataReq==0x01) || (DataReq==0xFF))
1386            {
1387                if(MTR1_MODE3 & PosDecMsk)  printf("E1: %08Hd\r\n", E1);
1388                else                        printf("E1: %06HX\r\n", E1);
1389            }
1390            if((DataReq==0x02) || (DataReq==0xFF))
1391            {
1392                if(MTR2_MODE3 & PosDecMsk)  printf("E2: %08Hd\r\n", E2);
1393                else                        printf("E2: %06HX\r\n", E2);
1394            }
1395        } // If TE
1396        //-------------------------------
1397        else if(CmdSource == I2C2_SrcMsk)
1398        { // I2C2 Mode
1399            if(DataReq==0xFF)
1400            { // Send Both Motor Positions; Mtr1 first.
1401                PktLen=9;
1402                Pkt[0]='E';
1403                Pkt[1]=0x06;
1404                Pkt[2]=(E1 & 0xFF);
1405                Pkt[3]=((E1>>8) & 0xFF);
1406                Pkt[4]=((E1>>16) & 0xFF);
1407                Pkt[5]=(E2 & 0xFF);
1408                Pkt[6]=((E2>>8) & 0xFF);
1409                Pkt[7]=((E2>>16) & 0xFF);
1410                SendI2C2Pkt();              // Transmit Pkt[].
1411            }
1412            else
1413            { // Single Motor request
1414                if(DataReq==0x01) E=E1; else E=E2;
1415                PktLen=6;
1416                Pkt[0]='E';
1417                Pkt[1]=0x03;
1418                Pkt[2]=(E & 0xFF);
1419                Pkt[3]=((E>>8) & 0xFF);
1420                Pkt[4]=((E>>16) & 0xFF);
1421                SendI2C2Pkt();              // Transmit Pkt[].
1422            }
1423        } // If I2C2
1424    }
1425    //-------------------------------------------------------------------------
1426    void DispV(void)
1427    {
1428        short long  V;
1429        long temp1,temp2,velRPM;
1430
1431        DispSvc &= ~VreqMsk;        // Clear request flag
1432
1433        if(CmdSource == API_SrcMsk)
1434        { // API Mode
1435            if(DataReq==0xFF)
1436            { // Send Both Motor Velocities; Mtr1 first.
```

```
1437                PktLen=12;
1438                Pkt[0]=chr_STX;
1439                Pkt[1]=HOST_NID;
1440                Pkt[2]='V';
1441                Pkt[3]=0x06;
1442                Pkt[4]=(V1 & 0xFF);
1443                Pkt[5]=((V1>>8) & 0xFF);
1444                Pkt[6]=((V1>>16) & 0xFF);
1445                Pkt[7]=(V2 & 0xFF);
1446                Pkt[8]=((V2>>8) & 0xFF);
1447                Pkt[9]=((V2>>16) & 0xFF);
1448                Pkt[11]=chr_ETX;
1449                SendApiPkt();
1450            }
1451            else
1452            { // Single Motor request
1453                if(DataReq==0x01) V=V1; else V=V2;
1454                PktLen=9;
1455                Pkt[0]=chr_STX;
1456                Pkt[1]=HOST_NID;
1457                Pkt[2]='V';
1458                Pkt[3]=0x03;
1459                Pkt[4]=(V & 0xFF);
1460                Pkt[5]=((V>>8) & 0xFF);
1461                Pkt[6]=((V>>16) & 0xFF);
1462                Pkt[8]=chr_ETX;
1463                SendApiPkt();
1464            }
1465        } // If API
1466        //-------------------------
1467        else if(CmdSource == TE_SrcMsk)
1468        { // TE Mode
1469    ////////////////////////////////////////////////////////////////////////
1470    //  Terminal Emulator interface velocity is shown in two units:       //
1471    //                      Ticks/VSP(HEX; 24 bits)                       //
1472    //                     RPM(Decimal)                                   //
1473    //                                                                    //
1474    //  V(RPM) = V(Ticks/VSP)*[1/VSP](VSP/ms)* 60000(ms/MIN)*[1/TPR](rev/Ticks) //
1475    ////////////////////////////////////////////////////////////////////////
1476            if( (DataReq==0x01) || (DataReq==0xFF) )
1477            { // If Motor 1 or both
1478                temp1 = (long)V1 * 60000;
1479                temp2 = (long)TPR1 * VSP1;
1480                velRPM = temp1/temp2;
1481                if(MTR1_MODE3 & VelDecMask)
1482                    printf("V1: %08Hd (%5ld rpm)", V1, velRPM);
1483                else printf("V1: %06HX (%5ld rpm)", V1, velRPM);
1484                if(vel1)    //If we are in ROVIM_T2D_MODE, print additional info
1485                    printf(" (%5ld Km/10/h)\r\n", vel1);
1486                else printf("\r\n");
1487            }
1488            if( (DataReq==0x02) || (DataReq==0xFF) )
1489            { // If Motor 2 or both
1490                temp1 = (long)V2 * 60000;
1491                temp2 = (long)TPR2 * VSP2;
1492                velRPM = temp1/temp2;
1493                if(MTR2_MODE3 & VelDecMask)
1494                    printf("V2: %08Hd (%5ld rpm)\r\n", V2,velRPM);
1495                else printf("V2: %06HX (%5ld rpm)\r\n", V2,velRPM);
1496
1497            }
1498        } // If TE
1499        //-------------------------
1500        else if(CmdSource == I2C2_SrcMsk)
1501        { // I2C2 Mode
1502            if(DataReq==0xFF)
1503            { // Send Both Motor Velocities; Mtr1 first.
1504                PktLen=9;
1505                Pkt[0]='V';
1506                Pkt[1]=0x06;
1507                Pkt[2]=(V1 & 0xFF);
1508                Pkt[3]=((V1>>8) & 0xFF);
1509                Pkt[4]=((V1>>16) & 0xFF);
```

```
1510                Pkt[5]=(V2 & 0xFF);
1511                Pkt[6]=((V2>>8) & 0xFF);
1512                Pkt[7]=((V2>>16) & 0xFF);
1513                SendI2C2Pkt();              // Transmit Pkt[].
1514
1515            }
1516            else
1517            { // Single Motor request
1518                if(DataReq==0x01) V=V1; else V=V2;
1519                PktLen=6;
1520                Pkt[0]='V';
1521                Pkt[1]=0x03;
1522                Pkt[2]=(V & 0xFF);
1523                Pkt[3]=((V>>8) & 0xFF);
1524                Pkt[4]=((V>>16) & 0xFF);
1525                SendI2C2Pkt();              // Transmit Pkt[].
1526            }
1527        } // If I2C2
1528    }
1529    //-------------------------------------------------------------------------
1530    void DispADC(void)  // Get ADC (Ch:0 .. Ch:6)
1531    {
1532        BYTE i;
1533        WORD AdcVal;                // ADC voltage in mV.
1534
1535        DispSvc &= ~ADCreqMsk;      // Clear request flag
1536
1537        if(CmdSource == API_SrcMsk)
1538        { // API Mode.  Transmit raw readings
1539            if(DataReq==0xFF)
1540            { // Send All Channnels
1541                PktLen=13;
1542                Pkt[0]=chr_STX;
1543                Pkt[1]=HOST_NID;
1544                Pkt[2]='C';
1545                Pkt[3]=0x07;
1546                for(i=0; i<7;i++)  Pkt[i+4] = ADC0[i];
1547                Pkt[12]=chr_ETX;
1548                SendApiPkt();
1549            }
1550            else
1551            { // Single Channel
1552                PktLen=7;
1553                Pkt[0]=chr_STX;
1554                Pkt[1]=HOST_NID;
1555                Pkt[2]='C';
1556                Pkt[3]=0x01;
1557                Pkt[4]=ADC0[DataReq];
1558                Pkt[6]=chr_ETX;
1559                SendApiPkt();
1560            }
1561        } // If API
1562        //-------------------------------
1563        else if(CmdSource == TE_SrcMsk)
1564        { // TE Mode.  Transmit readings in mV
1565            if(DataReq == 0xFF)
1566            { // Disp All Channels
1567                for(i=0; i<7; i++)
1568                {
1569                    AdcVal=AdcConvert((WORD)ADC0[i]);
1570                    printf("AN%1d: = %5d mV\r\n",i,AdcVal);
1571                }
1572                // Special case: vbatt (converted voltage from ADC0[6])
1573                printf("BAT: = %5d mV\r\n",AdcToVbatt(AdcVal));
1574            }
1575            else
1576            {
1577                AdcVal=AdcConvert((WORD)ADC0[DataReq]);
1578                printf("AN%1d: = %5d mV\r\n",DataReq,AdcVal);
1579            }
1580        } // If TE
1581        //-------------------------------
1582        else if(CmdSource == I2C2_SrcMsk)
```

```
1583      { // I2C2 Mode
1584          if(DataReq==0xFF)
1585          { // Disp All Channels.
1586              PktLen=10;
1587              Pkt[0]='C';
1588              Pkt[1]=0x07;
1589              for (i=0; i<7;i++)  Pkt[i+2] = ADC0[i];
1590              SendI2C2Pkt();              // Transmit Pkt[].
1591          }
1592          else
1593          { // Single Channel
1594              PktLen=4;
1595              Pkt[0]='C';
1596              Pkt[1]=0x01;
1597              Pkt[2]=ADC0[DataReq];
1598              SendI2C2Pkt();              // Transmit Pkt[].
1599          }
1600      } // If I2C2
1601  }
1602  //---------------------------------------------------------------------------
1603  void DispRC(void)
1604  {
1605      BYTE i;
1606      WORD j;
1607
1608      DispSvc &= ~RCreqMsk;         // Clear request flag
1609
1610      if(CmdSource == API_SrcMsk)
1611      { // API Mode
1612          if(DataReq==0xFF)
1613          { // Send All Channnels
1614              PktLen=12;
1615              Pkt[0]=chr_STX;
1616              Pkt[1]=HOST_NID;
1617              Pkt[2]='N';
1618              Pkt[3]=0x06;
1619              for (i=0; i<3;i++)
1620              {
1621                  j=RC[i];
1622                  Pkt[2*i+4] = j & 0xFF;
1623                  Pkt[2*i+5] = j >> 8;
1624              }
1625              Pkt[11]=chr_ETX;
1626              SendApiPkt();
1627          }
1628          else
1629          { // Single Channel
1630              PktLen=8;
1631              j=RC[DataReq-1];
1632              Pkt[0]=chr_STX;
1633              Pkt[1]=HOST_NID;
1634              Pkt[2]='N';
1635              Pkt[3]=0x02;
1636              Pkt[4] = j & 0xFF;
1637              Pkt[5] = j >> 8;
1638              Pkt[7]=chr_ETX;
1639              SendApiPkt();
1640          }
1641      } // If API
1642      //-------------------------------
1643      else if(CmdSource == TE_SrcMsk)
1644      { // TE Mode
1645          if(DataReq == 0xFF)
1646          { // All Channels
1647              for(i=0; i<=2; i++)
1648                  printf("Ch%1u: %5u uS (%03X)\r\n", i+1, RC[i],RC[i]);
1649          }
1650          else
1651          { // Single Channel
1652              i=DataReq-1;
1653              printf("Ch%1u: %5u uS (%03X)\r\n", DataReq,RC[i],RC[i]);
1654          }
1655      } // If TE
```

```
1656      //-------------------------------
1657      else if(CmdSource == I2C2_SrcMsk)
1658      { // I2C2 Mode
1659          if(DataReq==0xFF)
1660          { // All Channels.
1661              PktLen=9;
1662              Pkt[0]='N';
1663              Pkt[1]=0x06;
1664              for (i=0; i<3;i++)
1665              {
1666                  j=RC[i];
1667                  Pkt[2*i+2] = j & 0xFF;
1668                  Pkt[2*i+3] = j >> 8;
1669              }
1670              SendI2C2Pkt();              // Transmit Pkt[].
1671          }
1672          else
1673          { // Single Channel
1674              PktLen=5;
1675              Pkt[0]='N';
1676              Pkt[1]=0x02;
1677              i=DataReq-1;
1678              Pkt[2]=(RC[i] & 0xFF);
1679              Pkt[3]=(RC[i] >> 8);
1680              SendI2C2Pkt();              // Transmit Pkt[].
1681          }
1682      } // If I2C2
1683  }
1684  //-----------------------------------------------------------------------------
1685  void    DispMEM(void)               // Single Memory Byte.
1686  {
1687      DispSvc &= ~MEMBYTEreqMsk;  // Clear request flag
1688      if(CmdSource == API_SrcMsk)
1689      { // API Mode
1690          PktLen=7;
1691          Pkt[0]=chr_STX;
1692          Pkt[1]=HOST_NID;
1693          Pkt[2]='R';
1694          Pkt[3]=0x01;
1695          Pkt[4]=xbyte;
1696          Pkt[6]=chr_ETX;
1697          SendApiPkt();
1698      } // If API
1699      //-------------------------------
1700      else if(CmdSource == TE_SrcMsk)
1701      { // TE Mode
1702          printf("%02X\r\n", xbyte);
1703      }
1704      //-------------------------------
1705      else if(CmdSource == I2C2_SrcMsk)
1706      { // I2C2 Mode
1707          PktLen=4;
1708          Pkt[0]='R';
1709          Pkt[1]=0x01;
1710          Pkt[2]=xbyte;
1711          SendI2C2Pkt();              // Transmit Pkt[].
1712      } // If I2C2
1713  }
1714  //-----------------------------------------------------------------------------
1715  void    DispIO(void)                // Single IOEXP register.
1716  {
1717      DispSvc &= ~IOBYTEreqMsk;   // Clear request flag
1718      if(CmdSource == API_SrcMsk)
1719      { // API Mode
1720          PktLen=7;
1721          Pkt[0]=chr_STX;
1722          Pkt[1]=HOST_NID;
1723          Pkt[2]='K';
1724          Pkt[3]=0x01;
1725          Pkt[4]=xbyte;
1726          Pkt[6]=chr_ETX;
1727          SendApiPkt();
1728      } // If API
```

```
1729      //-------------------------------
1730      else if(CmdSource == TE_SrcMsk)
1731      { // TE Mode
1732          printf("%02X\r\n", xbyte);
1733      } // If TE
1734      //-------------------------------
1735      else if(CmdSource == I2C2_SrcMsk)
1736      { // I2C2 Mode
1737          PktLen=4;
1738          Pkt[0]='K';
1739          Pkt[1]=0x01;
1740          Pkt[2]=xbyte;
1741          SendI2C2Pkt();              // Transmit Pkt[].
1742      } // If I2C2
1743  }
1744  //-------------------------------------------------------------------------
1745  void    DispSTAT(void)          // Status
1746  {
1747      DispSvc &= ~STATreqMsk;     // Clear request flag
1748
1749      if(CmdSource == API_SrcMsk)
1750      { // API Mode
1751          if(DataReq == 0xFF)
1752          { // Both motors
1753              PktLen=18;
1754              Pkt[0]=chr_STX;
1755              Pkt[1]=HOST_NID;
1756              Pkt[2]='U';
1757              Pkt[3]=0x0C;
1758
1759              Pkt[4]=MTR1_MODE1;
1760              Pkt[5]=MTR1_MODE2;
1761              Pkt[6]=MTR1_MODE3;
1762              Pkt[7]=Power1;
1763              Pkt[8]=Mtr1_Flags1;
1764              Pkt[9]=Mtr1_Flags2;
1765
1766              Pkt[10]=MTR2_MODE1;
1767              Pkt[11]=MTR2_MODE2;
1768              Pkt[12]=MTR2_MODE3;
1769              Pkt[13]=Power2;
1770              Pkt[14]=Mtr2_Flags1;
1771              Pkt[15]=Mtr2_Flags2;
1772
1773              Pkt[17]=chr_ETX;
1774              SendApiPkt();
1775          }
1776          else
1777          { // Single motor
1778              if(DataReq == 0x01)
1779              { // Mtr1
1780                  PktLen=12;
1781                  Pkt[0]=chr_STX;
1782                  Pkt[1]=HOST_NID;
1783                  Pkt[2]='U';
1784                  Pkt[3]=0x06;
1785
1786                  Pkt[4]=MTR1_MODE1;
1787                  Pkt[5]=MTR1_MODE2;
1788                  Pkt[6]=MTR1_MODE3;
1789                  Pkt[7]=Power1;
1790                  Pkt[8]=Mtr1_Flags1;
1791                  Pkt[9]=Mtr1_Flags2;
1792
1793                  Pkt[11]=chr_ETX;
1794                  SendApiPkt();
1795              }
1796              else
1797              { // Mtr2
1798                  PktLen=12;
1799                  Pkt[0]=chr_STX;
1800                  Pkt[1]=HOST_NID;
1801                  Pkt[2]='U';
```

```
1802                    Pkt[3]=0x06;
1803
1804                    Pkt[4]=MTR2_MODE1;
1805                    Pkt[5]=MTR2_MODE2;
1806                    Pkt[6]=MTR2_MODE3;
1807                    Pkt[7]=Power2;
1808                    Pkt[8]=Mtr2_Flags1;
1809                    Pkt[9]=Mtr2_Flags2;
1810
1811                    Pkt[11]=chr_ETX;
1812                    SendApiPkt();
1813                }
1814            }
1815        } // If API
1816        //-------------------------------
1817        else if(CmdSource == TE_SrcMsk)
1818        { // TE Mode
1819            if( (DataReq==0x01) || (DataReq == 0xFF) )  MtrStatTE(0x01);
1820            if( (DataReq==0x02) || (DataReq == 0xFF) )  MtrStatTE(0x02);
1821        } // If TE
1822        //-------------------------------
1823        else if(CmdSource == I2C2_SrcMsk)
1824        { // I2C2 Mode
1825            if(DataReq == 0xFF)
1826            { // Both motors
1827                PktLen=15;
1828                Pkt[0]='U';
1829                Pkt[1]=12;
1830                Pkt[2]=MTR1_MODE1;
1831                Pkt[3]=MTR1_MODE2;
1832                Pkt[4]=MTR1_MODE3;
1833                Pkt[5]=Power1;
1834                Pkt[6]=Mtr1_Flags1;
1835                Pkt[7]=Mtr1_Flags2;
1836
1837                Pkt[8]=MTR2_MODE1;
1838                Pkt[9]=MTR2_MODE2;
1839                Pkt[10]=MTR2_MODE3;
1840                Pkt[11]=Power2;
1841                Pkt[12]=Mtr2_Flags1;
1842                Pkt[13]=Mtr2_Flags2;
1843
1844                SendI2C2Pkt();              // Transmit Pkt[].
1845            }
1846            else
1847            { // Single motor
1848                if(DataReq == 0x01)
1849                { // Mtr1
1850                    PktLen=9;
1851                    Pkt[0]='U';
1852                    Pkt[1]=0x06;
1853                    Pkt[2]=MTR1_MODE1;
1854                    Pkt[3]=MTR1_MODE2;
1855                    Pkt[4]=MTR1_MODE3;
1856                    Pkt[5]=Power1;
1857                    Pkt[6]=Mtr1_Flags1;
1858                    Pkt[7]=Mtr1_Flags2;
1859                    SendI2C2Pkt();              // Transmit Pkt[].
1860                }
1861                else
1862                { // Mtr2
1863                    PktLen=9;
1864                    Pkt[0]='U';
1865                    Pkt[1]=0x06;
1866                    Pkt[2]=MTR2_MODE1;
1867                    Pkt[3]=MTR2_MODE2;
1868                    Pkt[4]=MTR2_MODE3;
1869                    Pkt[5]=Power2;
1870                    Pkt[6]=Mtr2_Flags1;
1871                    Pkt[7]=Mtr2_Flags2;
1872                    SendI2C2Pkt();              // Transmit Pkt[].
1873                }
1874            } // If Single motor
```

```
1875      } // If I2C2
1876  }
1877  //-------------------------------------------------------------------------
1878  void      MtrStatTE(BYTE mtr)
1879  { // Utility used to display motor status
1880      BYTE m1, m2, m3, s, flg2;
1881
1882      if(mtr==1)
1883          { m1=MTR1_MODE1;m2=MTR1_MODE2;m3=MTR1_MODE3;s=Power1;flg2=Mtr1_Flags2; }
1884      else
1885          { m1=MTR2_MODE1;m2=MTR2_MODE2;m3=MTR2_MODE3;s=Power2;flg2=Mtr2_Flags2; }
1886
1887      // MOTOR ID
1888      printf("MTR#   :%1X\r\n",mtr);
1889
1890      // CONTROL MODE
1891      printf("MODE   :");
1892      if(m2 & PotcMsk) printf("POTC\r\n");
1893      else if (m2 & PotfMsk) printf("POTF\r\n");
1894      else if (m2 & RcNrmMsk) printf("RC\r\n");
1895      else if (m2 & RcMixMsk) printf("RCMIX\r\n");
1896      else if (m2 & RcServoMsk) printf("RCSVO\r\n");
1897      else if (m2 & PotServoMsk) printf("POTSVO\r\n");
1898      else if (m2 & PotMixMsk) printf("POTMIX\r\n");
1899      else printf("CMD\r\n");
1900
1901      // PWM DUTY CYCLE & DIRECTION
1902      if (flg2 & DisableMsk) printf("PWM    :DISABLED\r\n");
1903      else if (s==0) printf("PWM    :OFF\r\n");
1904      else if(flg2 & _MtrD_mask) printf("PWM    :REV(%3u%%)\r\n",s);
1905      else printf("PWM    :FWD(%3u%%)\r\n",s);
1906
1907      // TRIGGER MODE
1908      if(m1 & trigMsk) printf("TRIG   :ON\r\n");
1909      else printf("TRIG   :OFF\r\n");
1910
1911      // PID ERR SUMMATION HOLDOFF
1912      if(m1 & sumhoMsk) printf("SUMHO :ON\r\n");
1913      else printf("SUMHO :OFF\r\n");
1914
1915      // TARGET SPECIFICATION
1916      if(m1 & relMsk) printf("TGT    :REL\r\n");
1917      else printf("TGT    :ABS\r\n");
1918
1919      // OVER CURRENT
1920      if (m3 & OcieMsk)
1921      {
1922          if (m3 & OcFastOffMsk) printf("OCMODE:FASTOFF\r\n");
1923          else printf("OCMODE:SLOWRUN\r\n");
1924      }
1925      else printf("OCMODE:NONE\r\n");
1926
1927      printf("\r\n");
1928  }
1929  //-------------------------------------------------------------------------
1930  void DispPID()        // Motor PID runtime parameters
1931  {
1932      WORD kp,ki,kd;
1933      BYTE vsp,vmin,vmax;
1934
1935      //////////////////////////////////////////////////////////////
1936      // Outputs motor specific PID parameters as well as a few that  //
1937      // apply to both motors.                                        //
1938      //////////////////////////////////////////////////////////////
1939      DispSvc &= ~PIDreqMsk;        // Clear request flag
1940      if(DataReq==1)  {kp=KP1; ki=KI1; kd=KD1; vsp=VSP1; vmin=VMIN1; vmax=VMAX1;}
1941      else            {kp=KP2; ki=KI2; kd=KD2; vsp=VSP2; vmin=VMIN2; vmax=VMAX2;}
1942          //-----------------------------
1943      if(CmdSource == API_SrcMsk)
1944      { // API Mode
1945          PktLen=19;
1946          Pkt[0]=chr_STX;
1947          Pkt[1]=HOST_NID;
```

```
1948            Pkt[2]='P';
1949            Pkt[3]=0x0D;
1950            Pkt[4] = (kp & 0xFF);
1951            Pkt[5] = (kp >> 8);
1952            Pkt[6] = (ki & 0xFF);
1953            Pkt[7] = (ki >> 8);
1954            Pkt[8] = (kd & 0xFF);
1955            Pkt[9] = (kd >> 8);
1956            Pkt[10] = vsp;
1957            Pkt[11] = vmin;
1958            Pkt[12] = vmax;
1959            Pkt[13] = (MAXERR & 0xFF);
1960            Pkt[14] = (MAXERR >> 8);
1961            Pkt[15] = (MAXSUM & 0xFF);
1962            Pkt[16] = (MAXSUM >> 8);
1963            Pkt[18] = chr_ETX;
1964            SendApiPkt();
1965        } // If API
1966        //-------------------------------
1967        else if(CmdSource == TE_SrcMsk)
1968        { // TE Mode
1969            printf("MTR# :%1X\r\n",DataReq);
1970            printf("KP   :%04X\r\n",kp);
1971            printf("KI   :%04X\r\n",ki);
1972            printf("KD   :%04X\r\n",kd);
1973            printf("VSP  :%02X\r\n",vsp);
1974            printf("VMIN :%02X\r\n",vmin);
1975            printf("VMAX :%02X\r\n",vmax);
1976            printf("MAXE :%04X\r\n",MAXERR);
1977            printf("MAXS :%04X\r\n",MAXSUM);
1978        } // If TE
1979        //-------------------------------
1980        else if(CmdSource == I2C2_SrcMsk)
1981        { // I2C2 Mode
1982            PktLen=16;
1983            Pkt[0]='P';
1984            Pkt[1]=13;
1985            Pkt[2] = (kp & 0xFF);
1986            Pkt[3] = (kp >> 8);
1987            Pkt[4] = (ki & 0xFF);
1988            Pkt[5] = (ki >> 8);
1989            Pkt[6] = (kd & 0xFF);
1990            Pkt[7] = (kd >> 8);
1991            Pkt[8] = vsp;
1992            Pkt[9] = vmin;
1993            Pkt[10] = vmax;
1994            Pkt[11] = (MAXERR & 0xFF);
1995            Pkt[12] = (MAXERR >> 8);
1996            Pkt[13] = (MAXSUM & 0xFF);
1997            Pkt[14] = (MAXSUM >> 8);
1998            SendI2C2Pkt();              // Transmit Pkt[].
1999        } // If I2C2
2000    }
2001    //-------------------------------------------------------------------------------
2002    void    DispHMS(void)           // HH:MM:SS (RTC Time)
2003    {
2004        DispSvc &= ~HMSreqMsk;      // Clear request flag
2005        //-------------------------------
2006        if(CmdSource == API_SrcMsk)
2007        { // API Mode
2008            PktLen=9;
2009            Pkt[0]=chr_STX;
2010            Pkt[1]=HOST_NID;
2011            Pkt[2]='D';
2012            Pkt[3]=0x03;
2013            Pkt[4] = HOURS;
2014            Pkt[5] = MINS;
2015            Pkt[6] = SECS;
2016            Pkt[8]=chr_ETX;
2017            SendApiPkt();
2018        }
2019        //-------------------------------
2020        else if(CmdSource == TE_SrcMsk)
```

```
2021      { // TE Mode
2022          printf("%02u:%02u:%02u\r\n", HOURS, MINS, SECS);
2023      }
2024      //-------------------------------
2025      else if(CmdSource == I2C2_SrcMsk)
2026      { // I2C2 Mode
2027          PktLen=6;
2028          Pkt[0]='D';
2029          Pkt[1]=0x03;
2030          Pkt[2] = HOURS;
2031          Pkt[3] = MINS;
2032          Pkt[4] = SECS;
2033          SendI2C2Pkt();              // Transmit Pkt[].
2034      } // If I2C2
2035  }
2036  //-------------------------------------------------------------------------------
2037  void    DispExtEEBLOCK(void)    // Block of External EEPROM starting at 'pBYTE'
2038  {
2039      WORD    i,j;
2040      BYTE    hi,low;
2041      WORD    adrs;
2042
2043      DispSvc &= ~EXTEEBLKreqMsk; // Clear request flag
2044      adrs=(WORD)pBYTE;
2045      //-------------------------------
2046      if(CmdSource == API_SrcMsk)
2047      { // API Mode; Block Length variable (<= 0x80).
2048          PktLen=BlkLen+6;
2049          Pkt[0]=chr_STX;
2050          Pkt[1]=HOST_NID;
2051          Pkt[2]='L';
2052          Pkt[3]=BlkLen;
2053          for (i=4; i<BlkLen+4; i++)
2054          {
2055              j=ReadExtEE_Byte(adrs); adrs++;
2056              Pkt[i]=j;
2057          }
2058          Pkt[BlkLen+5]=chr_ETX;
2059          SendApiPkt();
2060      } // If API
2061      //-------------------------------
2062      else if(CmdSource == TE_SrcMsk)
2063      { // TE Mode; Block Length fixed (0x80);  Display as paired bytes.
2064
2065          printf("External EEPROM\r\n");
2066          for (i=0; i<128; i+=8)
2067          {
2068              printf("%04X:",adrs);
2069              for (j=0; j<8; j+=2)
2070              {
2071                  low=ReadExtEE_Byte(adrs); adrs++;
2072                  hi=ReadExtEE_Byte(adrs);  adrs++;
2073                  printf("%02X%02X ",low,hi);
2074              }
2075              printf("\r\n");
2076          }
2077          printf("\r\n");
2078      } // If TE
2079      //-------------------------------
2080      else if(CmdSource == I2C2_SrcMsk)
2081      { // I2C2 Mode
2082          PktLen=BlkLen+3;
2083          Pkt[0]='L';
2084          Pkt[1]=BlkLen;
2085          for (i=2; i<BlkLen+2; i++)
2086          {
2087              j=ReadExtEE_Byte(adrs); adrs++;
2088              Pkt[i]=j;
2089          }
2090          SendI2C2Pkt();              // Transmit Pkt[].
2091      } // If I2C2
2092  }
2093  //-------------------------------------------------------------------------------
```

```
2094 void    DispRAMBLOCK(void)      // 128 bytes of RAM starting at 'pBYTE'
2095 {
2096     WORD    i,j;
2097     BYTE    hi,low;
2098
2099     DispSvc &= ~RAMBLKreqMsk;   // Clear request flag
2100     //-------------------------------
2101     if(CmdSource == API_SrcMsk)
2102     { // API Mode (BlkLen: Block Length <= 0x80)
2103         PktLen=BlkLen+6;
2104         Pkt[0]=chr_STX;
2105         Pkt[1]=HOST_NID;
2106         Pkt[2]='L';
2107         Pkt[3]=BlkLen;
2108         for (i=4; i<BlkLen+4; i++)
2109         {
2110             j=*pBYTE; pBYTE++;
2111             Pkt[i]=j;
2112         }
2113         Pkt[BlkLen+5]=chr_ETX;
2114         SendApiPkt();
2115     } // If API
2116     //-------------------------------
2117     else if(CmdSource == TE_SrcMsk)
2118     { // TE Mode (Block Length assumed 0x80)
2119         printf("RAM\r\n");
2120         for (i=0; i<128; i+=8)
2121         {
2122             printf("%04X:",pBYTE);
2123             for (j=0; j<8; j+=2)
2124             {
2125                 low=*pBYTE; pBYTE++; hi=*pBYTE; pBYTE++;
2126                 printf("%02X%02X ",low,hi);
2127             }
2128             printf("\r\n");
2129         }
2130         printf("\r\n");
2131     } // If TE
2132     //-------------------------------
2133     else if(CmdSource == I2C2_SrcMsk)
2134     { // I2C2 Mode
2135         PktLen=BlkLen+3;
2136         Pkt[0]='L';
2137         Pkt[1]=BlkLen;
2138         for (i=2; i<BlkLen+2; i++)
2139         {
2140             j=*pBYTE; pBYTE++;
2141             Pkt[i]=j;
2142         }
2143         SendI2C2Pkt();              // Transmit Pkt[].
2144     } // If I2C2
2145 }
2146 //-------------------------------------------------------------------------------
2147 void    DispIntEEBLOCK(void)    // Block of Internal EEPROM starting at 'pBYTE'
2148 {
2149     WORD    i,j;
2150     BYTE    hi,low;
2151     WORD    adrs;
2152
2153     DispSvc &= ~INTEEBLKreqMsk; // Clear request flag
2154     adrs=(WORD)pBYTE;
2155     //-------------------------------
2156     if(CmdSource == API_SrcMsk)
2157     { // API Mode (BlkLen: Block Length <= 0x80)
2158         PktLen=BlkLen+6;
2159         Pkt[0]=chr_STX;
2160         Pkt[1]=HOST_NID;
2161         Pkt[2]='L';
2162         Pkt[3]=BlkLen;
2163         for (i=4; i<BlkLen+4; i++)
2164         {
2165             j=ReadIntEE_Byte(adrs); adrs++;
2166             Pkt[i]=j;
```

```
2167            }
2168            Pkt[BlkLen+5]=chr_ETX;
2169            SendApiPkt();
2170        } // If API
2171        //-------------------------------
2172        else if(CmdSource == TE_SrcMsk)
2173        { // TE Mode (Block Length assumed 0x80)
2174
2175            printf("Internal EEPROM\r\n");
2176            for (i=0; i<128; i+=8)
2177            {
2178                printf("%04X:",adrs);
2179                for (j=0; j<8; j+=2)
2180                {
2181                    low=ReadIntEE_Byte(adrs); adrs++;
2182                    hi=ReadIntEE_Byte(adrs);  adrs++;
2183                    printf("%02X%02X ",low,hi);
2184                }
2185                printf("\r\n");
2186            }
2187            printf("\r\n");
2188        } // If TE
2189        //-------------------------------
2190        else if(CmdSource == I2C2_SrcMsk)
2191        { // I2C2 Mode
2192            PktLen=BlkLen+3;
2193            Pkt[0]='L';
2194            Pkt[1]=BlkLen;
2195            for (i=2; i<BlkLen+2; i++)
2196            {
2197                j=ReadIntEE_Byte(adrs); adrs++;
2198                Pkt[i]=j;
2199            }
2200            SendI2C2Pkt();              // Transmit Pkt[].
2201        } // If I2C2
2202    }
2203    //-----------------------------------------------------------------------------
2204    void    Tune1(void) // Mtr1 PID Tuning Aid: CmdQ Verbose output
2205    {
2206        BYTE i,index;
2207
2208        ////////////////////////////////////////////////////////////////////
2209        // Conditionally display PID Err for Mtr1.                        //
2210        //                                                                //
2211        //  Primary usage: See "PID TUNING" Cmd.                          //
2212        //     Does nothing unless all conditions are met:                //
2213        //       1) npid <= PidLimit.                                     //
2214        //       2) PID is active.                                        //
2215        //       3) Verbose mode active.                                  //
2216        //                                                                //
2217        // For efficiency, API packets contain 8 samples of Err.          //
2218        ////////////////////////////////////////////////////////////////////
2219        if( (Mtr1_Flags1 & pida_Msk) &&
2220            (npid1 < Pid1Limit) &&
2221            (MTR1_MODE3 & VerboseMsk))
2222        { // If something to do, ..
2223            //-------------------------
2224            if(CmdSource == API_SrcMsk)
2225            { // API Mode: Xmit in packets that contain 8 samples
2226                //
2227                // index: ptr to 3-byte Err field within DATA portion of pkt
2228                index = 4 + 3*(BYTE)(npid1 & 0x07);
2229
2230                npid1++;
2231                Pkt[index]=(Err1 & 0xFF);           // Low Byte
2232                Pkt[index+1]=((Err1 >> 8) & 0xFF);  // Mid Byte
2233                Pkt[index+2]=(Err1 >> 16);          // Hi Byte
2234
2235                // Special case: Test for partial last packet
2236                if( ( npid1 == Pid1Limit ) && ( index != 25 ) )
2237                {
2238                    for ( i = index; i <= 25; i+=3 )
2239                     { // Zero fill unused portion of DATA field
```

```
2240                    Pkt[i]=0; Pkt[i+1]=0; Pkt[i+2]=0;
2241                 }
2242                index = 25;
2243            }
2244
2245            if( index == 25 )
2246            { // Pkt full.  Time to send
2247                Pkt[0]=chr_STX;
2248                Pkt[1]=HOST_NID;
2249                Pkt[2]='Q';
2250                Pkt[3]=24;          // 8 Err samples (3 bytes per sample)
2251                Pkt[29]=chr_ETX;
2252                PktLen=30;
2253                SendApiPkt();
2254            }
2255        } // If API
2256        //-------------------------------
2257        else if(CmdSource == TE_SrcMsk)
2258        { // TE Mode: Xmit line to TE
2259            npid1++;
2260            if(npid1==1) printf("STEP RESPONSE:1\r\n");
2261            printf("%3d:%+6Hd\r\n", npid1,Err1);
2262        } // If TE
2263        //-------------------------------
2264        else if(CmdSource == I2C2_SrcMsk)
2265        { // I2C2 Mode: Xmit in packets that contain 8 samples.
2266            //
2267            // index: ptr to 3-byte Err field within DATA portion of pkt
2268            index = 2 + 3*(BYTE)(npid1 & 0x07);
2269
2270            npid1++;
2271            Pkt[index]=(Err1 & 0xFF);               // Low Byte
2272            Pkt[index+1]=((Err1 >> 8) & 0xFF);      // Mid Byte
2273            Pkt[index+2]=(Err1 >> 16);              // Hi Byte
2274
2275            // Special case: Test for partial last packet
2276            if( ( npid1 == Pid1Limit ) && ( index != 23 ) )
2277            {
2278                for ( i = index; i <= 23; i+=3 )
2279                { // Zero fill unused portion of DATA field
2280                    Pkt[i]=0; Pkt[i+1]=0; Pkt[i+2]=0;
2281                }
2282                index = 23;
2283            }
2284
2285            if( index == 23 )
2286            { // Pkt full.  Time to send
2287                Pkt[0]='Q';
2288                Pkt[1]=24;          // 8 Err samples (3 bytes per sample)
2289                PktLen=27;
2290                SendI2C2Pkt();      // Transmit Pkt[].
2291            }
2292        } // If I2C2
2293        //-------------------------------
2294    } // something to do
2295
2296    // Reset PidLimit after last output to inhibit in other cmds.
2297    if( npid1 == Pid1Limit ) Pid1Limit=0;
2298 }
2299 //-------------------------------------------------------------------------
2300 void    Tune2(void) // Mtr2 PID Tuning Aid: CmdQ Verbose output
2301 {
2302    BYTE i,index;
2303
2304    ////////////////////////////////////////////////////////////////
2305    // Conditionally display PID Err for Mtr2.                     //
2306    //                                                            //
2307    //  Primary usage: See "PID TUNING" Cmd.                      //
2308    //     Does nothing unless all conditions are met:            //
2309    //       1) npid <= PidLimit.                                 //
2310    //       2) PID is active.                                    //
2311    //       3) Verbose mode active.                              //
2312    //                                                            //
```

```
2313    // For efficiency, API packets contain 8 samples of Err.            //
2314    //////////////////////////////////////////////////////////////////////
2315    if( (Mtr2_Flags1 & pida_Msk) &&
2316        (npid2 < Pid2Limit) &&
2317        (MTR2_MODE3 & VerboseMsk))
2318    { // If something to do, ..
2319        //-----------------------------
2320        if(CmdSource == API_SrcMsk)
2321        { // API Mode: Xmit in packets that contain 8 samples
2322            //
2323            // index: ptr to 3-byte Err field within DATA portion of pkt
2324            index = 4 + 3*(BYTE)(npid2 & 0x07);
2325
2326            npid2++;
2327            Pkt[index]=(Err2 & 0xFF);               // Low Byte
2328            Pkt[index+1]=((Err2 >> 8) & 0xFF);      // Mid Byte
2329            Pkt[index+2]=(Err2 >> 16);              // Hi Byte
2330
2331            // Special case: Test for partial last packet
2332            if( ( npid2 == Pid2Limit ) && ( index != 25 ) )
2333            {
2334                for ( i = index; i <= 25; i+=3 )
2335                 { // Zero fill unused portion of DATA field
2336                    Pkt[i]=0; Pkt[i+1]=0; Pkt[i+2]=0;
2337                 }
2338                index = 25;
2339            }
2340
2341            if( index == 25 )
2342            { // Pkt full.  Time to send
2343                Pkt[0]=chr_STX;
2344                Pkt[1]=HOST_NID;
2345                Pkt[2]='Q';
2346                Pkt[3]=24;          // 8 Err samples (3 bytes per sample)
2347                Pkt[29]=chr_ETX;
2348                PktLen=30;
2349                SendApiPkt();
2350            }
2351        } // If API
2352        //-----------------------------
2353        else if(CmdSource == TE_SrcMsk)
2354        { // TE Mode: Xmit line to TE
2355            npid2++;
2356            if(npid2==1) printf("STEP RESPONSE:2\r\n");
2357            printf("%3d:%+6Hd\r\n", npid2,Err2);
2358        } // If TE
2359        //-----------------------------
2360        else if(CmdSource == I2C2_SrcMsk)
2361        { // I2C2 Mode: Xmit in packets that contain 8 samples.
2362            //
2363            // index: ptr to 3-byte Err field within DATA portion of pkt
2364            index = 2 + 3*(BYTE)(npid2 & 0x07);
2365
2366            npid2++;
2367            Pkt[index]=(Err2 & 0xFF);               // Low Byte
2368            Pkt[index+1]=((Err2 >> 8) & 0xFF);      // Mid Byte
2369            Pkt[index+2]=(Err2 >> 16);              // Hi Byte
2370
2371            // Special case: Test for partial last packet
2372            if( ( npid2 == Pid2Limit ) && ( index != 23 ) )
2373            {
2374                for ( i = index; i <= 23; i+=3 )
2375                { // Zero fill unused portion of DATA field
2376                    Pkt[i]=0; Pkt[i+1]=0; Pkt[i+2]=0;
2377                }
2378                index = 23;
2379            }
2380
2381            if( index == 23 )
2382            { // Pkt full.  Time to send
2383                Pkt[0]='Q';
2384                Pkt[1]=24;          // 8 Err samples (3 bytes per sample)
2385                PktLen=27;
```

```
2386                    SendI2C2Pkt();      // Transmit Pkt[].
2387                }
2388            } // If I2C2
2389            //-------------------------------
2390        } // something to do
2391
2392        // Reset PidLimit after last output to inhibit in other cmds.
2393        if( npid2 == Pid2Limit ) Pid2Limit=0;
2394    }
2395    //-----------------------------------------------------------------------
2396
2397
2398
2399
2400
2401
2402    //-----------------------------------------------------------------------
2403    void    ResetPIC(void)          // Reset Dalf Board
2404    {
2405        DispSvc &= ~RESETreqMsk;        // Clear request flag
2406        DoReset();                  // Microcontroller reset
2407    }
2408    //-----------------------------------------------------------------------
2409    void    WinkLEDS(void)          // Briefly blink programmable LED's
2410    {
2411        _LED1_ON;
2412        _LED2_ON;
2413        _LED3_ON;
2414        SetDelay(&Delay1, 0, msec_200); // 200 msec delay
2415        while( !Timeout(&Delay1) );
2416        _LED1_OFF;
2417        _LED2_OFF;
2418        _LED3_OFF;
2419    }
2420    //-----------------------------------------------------------------------
2421    void    InitLED(void)           // Initialization for ServiceLED()
2422    {
2423        ledshift = 0x80000000;
2424        ledcount = LED_PERIOD;
2425    }
2426    //-----------------------------------------------------------------------
2427    void    ServiceLED(void)        // Periodic LED service
2428    {
2429    ////////////////////////////////////////////////////////////////////////
2430    //      Four possible patterns on LED1 and LED2 indicating motor power:     //
2431    //                                                                          //
2432    //  OFF:        Power=0. No Power applied. ------------------ Off        //
2433    //  FAST BLINK: Power>0, TGA active.  Power applied. ------- TGA Active  //
2434    //  SLOW BLINK: Power>0, TGA inactive, Power applied, V>0. - Open loop move. //
2435    //  ON:         Power>0, TGA inactive, Power applied, V=0. - Stalled.    //
2436    //                                                                          //
2437    //                  Three possible patterns on LED3                     //
2438    //  OFF:            No Error.                                           //
2439    //  FAST BLINK:     Over Current (possibly transient condition)         //
2440    //  SLOW BLINK:     R/C signal loss (possibly transient condition)      //
2441    //  ON:             Low Batt (VBATT < VBWARN)                           //
2442    ////////////////////////////////////////////////////////////////////////
2443        // Reset counter and do right shift (with wrap) on led scheduler.
2444        ledshift >>= 1; if(!ledshift) ledshift = 0x80000000;
2445
2446        // Determine appropriate LED1 pattern
2447        if(!Power1)                             grn1pattern = LED_MTR_OFF;
2448        else if(Mtr1_Flags1 & tga_Msk)          grn1pattern = LED_MTR_TGA;
2449        else if(V1)                             grn1pattern = LED_MTR_OPENLP;
2450        else                                    grn1pattern = LED_MTR_STALL;
2451
2452        // Determine appropriate LED2 pattern
2453        if(!Power2)                             grn2pattern = LED_MTR_OFF;
2454        else if(Mtr2_Flags1 & tga_Msk)          grn2pattern = LED_MTR_TGA;
2455        else if(V2)                             grn2pattern = LED_MTR_OPENLP;
2456        else                                    grn2pattern = LED_MTR_STALL;
2457
2458        // Determine appropriate LED3 pattern
```

```
2459      if(LedErr==0)                          redpattern = LED_FULLOFF;
2460      else if(LedErr & (OC1msk + OC2msk))    redpattern = LED_OVERCURRENT;
2461      else if(LedErr & (SL1msk) + SL2msk)    redpattern = LED_SIGNAL_LOSS;
2462      else if(LedErr & VBATTmsk)             redpattern = LED_VBATT;
2463
2464
2465      // Adjust LED's as required.
2466      if(ledshift & grn1pattern) _LED1_ON; else _LED1_OFF;
2467      if(ledshift & grn2pattern) _LED2_ON; else _LED2_OFF;
2468      if(ledshift & redpattern)  _LED3_ON; else _LED3_OFF;
2469  }
2470  //-------------------------------------------------------------------------
2471  void    Motor1Current(void) // Voltage Window Transition - OverCurrent Sense
2472  {
2473      ////////////////////////////////////////////////////////////////////
2474      //  This code is part of the response to a transition of the      //
2475      //  current sense voltage into or out of the the voltage window   //
2476      //  defined by the digital current limiting pots.                 //
2477      //                                                                //
2478      //  The response depends on whether the transition is out of (over //
2479      //  current) or into (current acceptable) the window as well as   //
2480      //  the setting of the "oc_fastoff" bit.                          //
2481      //                                                                //
2482      //  If the "oc_fastoff" bit is set, and the transition is because //
2483      //  of overcurrent, most of the response will have already been   //
2484      //  handled directly by the ISR.  Power to the motor will already //
2485      //  have been removed and the motor control interface disabled.   //
2486      //  Recovery requires a board reset.                              //
2487      //                                                                //
2488      //  Otherwise, if the "oc_fastoff" bit is clear, the interface will //
2489      // "recover" when the current again becomes acceptable.           //
2490      //                                                                //
2491      //  The over current condition can be monitored by the state of   //
2492      //  Mtr1_Flags1.OverCurrent bit.                                  //
2493      ////////////////////////////////////////////////////////////////////
2494      if(ISR_Flags & OverC1Msk)       // Over current detected
2495      {
2496          // Flag over current
2497          Mtr1_Flags1 |= OverCurrent_Msk; // Flag over current condition
2498          LedErr |= OC1msk;               // Record also in LedErr
2499
2500          //  If "oc_fastoff"=1, just fix some mtr control flags and exit.
2501          if(MTR1_MODE3 & OcFastOffMsk)
2502          {
2503              Mtr1FlagFix();
2504              return;
2505          }
2506
2507          ////////////////////////////////////////////////////////
2508          // Schedule a motor stop by issuing a "Stop Motor Cmd"  //
2509          // with a compatible CmdSource.  Leave operating mode   //
2510          // unchanged.                                           //
2511          ////////////////////////////////////////////////////////
2512          if (MTR1_MODE2 & PotMsk) CmdSource = POT_SrcMsk;
2513          else if (MTR1_MODE2 & RcMsk) CmdSource = RC_SrcMsk;
2514          else CmdSource = TE_SrcMsk;
2515          SoftStop(0x01);
2516      }
2517      else                            // Current now ok
2518      {
2519          Mtr1_Flags1 &= ~OverCurrent_Msk; // Reset over current condition
2520          LedErr &= ~OC1msk;               // Reset also in LedErr
2521      }
2522  }
2523  //-------------------------------------------------------------------------
2524  void    Motor2Current(void) // Voltage Window Transition - OverCurrent Sense
2525  {
2526      ////////////////////////////////////////////////////////////////////
2527      //  This code is part of the response to a transition of the      //
2528      //  current sense voltage into or out of the the voltage window   //
2529      //  defined by the digital current limiting pots.                 //
2530      //                                                                //
2531      //  The response depends on whether the transition is out of (over //
```

```
2532      //  current) or into (current acceptable) the window as well as     //
2533      //  the setting of the "oc_fastoff" bit.                             //
2534      //                                                                   //
2535      //  If the "oc_fastoff" bit is set, and the transition is because    //
2536      //  of overcurrent, most of the response will have already been      //
2537      //  handled directly by the ISR.  Power to the motor will already    //
2538      //  have been removed and the motor control interface disabled.      //
2539      //  Recovery requires a board reset.                                 //
2540      //                                                                   //
2541      //  Otherwise, if the "oc_fastoff" bit is clear, the interface will  //
2542      // "recover" when the current again becomes acceptable.              //
2543      //                                                                   //
2544      //  The over current condition can be monitored by the state of      //
2545      //  Mtr2_Flags1.OverCurrent bit.                                     //
2546      ///////////////////////////////////////////////////////////////////////
2547      if(ISR_Flags & OverC2Msk)        // Over current detected
2548      {
2549          // Flag over current
2550          Mtr2_Flags1 |= OverCurrent_Msk;      // Flag over current condition
2551          LedErr |= OC2msk;                    // Record also in LedErr
2552
2553          //  If "oc_fastoff"=1, just fix some mtr control flags and exit.
2554          if(MTR2_MODE3 & OcFastOffMsk)
2555          {
2556              Mtr2FlagFix();
2557              return;
2558          }
2559
2560          ///////////////////////////////////////////////////////
2561          // Schedule a motor stop by issuing a "Stop Motor Cmd"   //
2562          // with a compatible CmdSource.  Leave operating mode    //
2563          // unchanged.                                            //
2564          ///////////////////////////////////////////////////////
2565          if (MTR2_MODE2 & PotMsk) CmdSource = POT_SrcMsk;
2566          else if (MTR2_MODE2 & RcMsk) CmdSource = RC_SrcMsk;
2567          else CmdSource = TE_SrcMsk;
2568          SoftStop(0x02);
2569      }
2570      else                             // Current now ok
2571      {
2572          Mtr2_Flags1 &= ~OverCurrent_Msk;     // Reset over current condition
2573          LedErr &= ~OC2msk;                   // Reset also in LedErr
2574      }
2575  }
2576  //---------------------------------------------------------------------------
2577
2578
2579
2580
2581
2582  //---------------------------------------------------------------------------
2583  void     GetApiPktCkSum(void)     // Compute and store Pkt CkSum Byte
2584  { // Entry: Pkt[], PktLen.  Exit: Pkt[PktLen-2]=CkSum
2585      BYTE cksum=0, i;
2586      Pkt[PktLen-2]=0;             // Pre-Fill CkSum field with zero.
2587      for (i=0; i<PktLen; i++) cksum+=Pkt[i];
2588      Pkt[PktLen-2] = ~cksum + 1;
2589  }
2590  //---------------------------------------------------------------------------
2591  void     GetI2C2PktCkSum(void)    // Compute and store Pkt CkSum Byte
2592  { // Entry: Pkt[], PktLen.  Exit: Pkt[PktLen-1]=CkSum
2593      BYTE cksum=0, i;
2594      Pkt[PktLen-1]=0;                         // Pre-Fill CkSum field with zero.
2595      for (i=0; i<PktLen; i++) cksum+=Pkt[i];
2596      Pkt[PktLen-1] = ~cksum + 1;
2597  }
2598  //---------------------------------------------------------------------------
2599  void     SendApiPkt(void)         // Transmit API Pkt to Host.
2600  { // Entry: Pkt[], PktLen.
2601      BYTE i;
2602      GetApiPktCkSum();
2603      for (i=0; i<PktLen; i++) printf("%c",Pkt[i]);
2604  }
```

```
2605 //-------------------------------------------------------------------------
2606 void     SendI2C2Pkt(void)        // Transmit I2C2 Pkt[] to Host
2607 {
2608     GetI2C2PktCkSum();              // Pkt[PktLen-1] <-- CKSUM.
2609     WriteSSP2BUF(Pkt[0]);           // Start Transmission
2610 }
2611 //-------------------------------------------------------------------------
2612 void     WriteSSP2BUF(BYTE chr)   // Load I2C2 Transmit Buffer
2613 {    //-----------------------------------------------------
2614     // If the SSP2 ISR has requested data, we start transmit of the
2615     // first char (Pkt[0]) here.  Subsequent bytes to transmit, are loaded by
2616     // the ISR starting at Pkt[1].
2617     //
2618     // Else, we flag that transmission hasn't yet started so the
2619     // ISR can auto-start transmission later from Pkt[0].
2620     //-----------------------------------------------------
2621     if(I2C2_State==3)
2622     { // Data requested by SSP2_ISR.  Start xmit now.
2623         SSP2BUF=chr;                    // Load Xmit Buffer
2624         SSP2CON1bits.CKP=1;             // Release SCLK (Start Xmit)
2625     }
2626     else
2627     { // Data has not yet been requested by SSP2_ISR.
2628         I2C2_PktStatus |= i2c2_AutoMsk;     // Flag data Pkt[] ready.  AutoStart
2629     }
2630 }
2631 //-------------------------------------------------------------------------
2632
2633
2634
2635
2636
2637
2638 //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2639 //+++          Interrupt Service Actions (called by Main Loop)           +++
2640 //+++                                                                    +++
2641 //+++   These services are the place for interrupt driven actions that   +++
2642 //+++   might introduce too much interrupt latency if included within the +++
2643 //+++   ISR handlers.                                                    +++
2644 //+++                                                                    +++
2645 //+++      ========================================================      +++
2646 //+++      The main loop calls SvcDispatch() with an index: 0<=i<=15     +++
2647 //+++      which is used to access routines Svc0, Svc1, ... SvcF.  The   +++
2648 //+++      ISR will have set the appropriate bit [0..F] in SERVICE_REQ.  +++
2649 //+++      ========================================================      +++
2650 //+++                                                                    +++
2651 //+++   Interrupts will be enabled during execution of these routines.   +++
2652 //+++                                                                    +++
2653 //+++   Some services are driven by timer interrupts which make regular  +++
2654 //+++   periodic requests for service (eg; TMR0 and TMR1 services).      +++
2655 //+++   Note that processing of the Svc's can be delayed by other tasks. +++
2656 //+++                                                                    +++
2657 //+++                                                                    +++
2658 //+++    SVC    INT       DESCRIPTION                          FREQ      +++
2659 //+++   ------  --------  -----------------------------------  --------  +++
2660 //+++   Svc0    TMR0      Mtr Command Processing               1 msec    +++
2661 //+++   Svc1    INT0      Mtr2 Encoder (AB2INT)                varies    +++
2662 //+++   Svc2    INT1      Mtr1 Encoder (AB1INT)                varies    +++
2663 //+++   Svc3    TMR1      RTC                                  1 sec     +++
2664 //+++   Svc4    TMR2      ADC State Machine (end of sequence)  35 msec (*)+++
2665 //+++   Svc5    INT2      Mtr2 Current Sense (I2INT)           varies    +++
2666 //+++   Svc6    INT3      Mtr1 Current Sense (I1INT)           varies    +++
2667 //+++   Svc7    ////      ///////                              //////    +++
2668 //+++   Svc8    TX1       USART1 Transmit                      varies    +++
2669 //+++   Svc9    RC1       USART1 Receive                       varies    +++
2670 //+++   SvcA    CCP1      EX0 - R/C Channel 1                  varies    +++
2671 //+++   SvcB    CCP2      EX1 - R/C Channel 2                  varies    +++
2672 //+++   SvcC    CCP3      EX3 - R/C Channel 3                  varies    +++
2673 //+++   SvcD    SSP2      I2C2 - Secondary I2C Bus (SLAVE)     varies    +++
2674 //+++   SvcE    ////      ///////                              //////    +++
2675 //+++   SvcF    ////      ///////                              //////    +++
2676 //+++                                                                    +++
2677 //+++     (*) - Timing affected by parameters in Parameter Block         +++
```

```
2678 //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
2679
2680
2681 //-------------------------------------------------------------------------------
2682 // Svc0 - TMR0 Service Request.                          Request every 1 msec.
2683 //-------------------------------------------------------------------------------
2684 void Svc0(void)          // TMR0: Heartbeat (1msec)
2685 {
2686     ////////////////////////////////////////////////////////////////////////
2687     //  Svc0 performs two types of tasks:                                 //
2688     //                                                                    //
2689     //  1) EXECUTE NEW COMMANDS                                           //
2690     //       Commands are received from serial (Svc9), I2C2, RC, and POTs.//
2691     //       Serial and I2C2 cmds are executed shortly after receipt. RC  //
2692     //       and POT cmds execute according to schedule maintained by     //
2693     //       the TIMESVC flags.  If enabled, serial cmd interface is checked//
2694     //       for timeout.                                                 //
2695     //                                                                    //
2696     //  2) MOTOR SERVICING                                                //
2697     //       Open loop commands: Service consists of ramp to desired speed.//
2698     //       Closed loop commands: Service consists of periodic calls to  //
2699     //       the Trajectory Generator and PID functions to directly alter //
2700     //       the motor PWM duty cycle.                                    //
2701     ////////////////////////////////////////////////////////////////////////
2702
2703     // Conditionally service on-board LED's
2704     ledcount--; if(!ledcount)
2705     {
2706         ledcount=LED_PERIOD;
2707         #ifdef DALF_ROVIM_T2D
2708         ROVIM_T2D_ServiceLED();
2709         #else //DALF_ROVIM_T2D
2710         ServiceLED();
2711         #endif //DALF_ROVIM_T2D
2712     }
2713     #ifdef DALF_ROVIM_T2D
2714     ROVIM_T2D_sysmonitorcount--;
2715     //For debug purposes, this task can be triggered by the user
2716     if((!ROVIM_T2D_sysmonitorcount) && (!ManualSysMonitoring))
2717     {
2718         ROVIM_T2D_sysmonitorcount=ROVIM_T2D_SYSTEM_MONITOR_PERIOD;
2719         ROVIM_T2D_MonitorSystem();
2720     }
2721     ROVIM_T2D_pwmrefreshcount--;
2722     if(!ROVIM_T2D_pwmrefreshcount)
2723     {
2724         ROVIM_T2D_pwmrefreshcount=ROVIM_T2D_PWM_REFRESH_PERIOD;
2725         ROVIM_T2D_ServicePWM();
2726     }
2727     #endif //DALF_ROVIM_T2D
2728
2729 #ifdef WATCHDOG_ENABLED
2730     watchdogcount--; if(!watchdogcount) {watchdogcount=WATCHDOG_PERIOD; KickWatchdog(); }
2731 #endif
2732 #ifdef DALF_TEST_ENABLED
2733     //TEST_Svc0();
2734 #endif
2735
2736     // Capture timed service requests.
2737     TIMESVC = TIMESVC_REQ;              // TIMESVC = Working copy
2738     TIMESVC_REQ ^= TIMESVC;             // Clear active Svc Req Flags
2739
2740     ////////////////////////////////////////////////////////////////////
2741     //      P E N D I N G    C M D    S E R V I C E S                  //
2742     //                                                                 //
2743     //  If motor now stopped (Power1=0 or Power2=0), and pending       //
2744     //  cmd is awaiting motor stopped, do it now.                      //
2745     ////////////////////////////////////////////////////////////////////
2746     PendingCmd1();
2747     PendingCmd2();
2748
2749     ////////////////////////////////////////////
2750     //     R A M P I N G      S E R V I C E S     //
```

```
2751      //////////////////////////////////////////////
2752      RampMotor1();             // Conditionally ramp mtr1
2753      RampMotor2();             // Conditionally ramp mtr2
2754
2755
2756      ////////////////////////////////////
2757      //    V S P 1    S E R V I C E S     //
2758      ////////////////////////////////////
2759      if(TIMESVC & Vsp1Msk)    // If time to sample Motor Velocity
2760      {
2761          #ifdef DALF_ROVIM_T2D
2762          ROVIM_T2D_UpdateVelocity1();  // Update Motor Velocity and Acceleration
2763          #else //DALF_ROVIM_T2D
2764          UpdateVelocity1();  // Update Motor Velocity
2765          #endif //DALF_ROVIM_T2D
2766          Trajectory1();        // Conditional mtr1 trajectory
2767          PID1();               // Conditional mtr1 PID Control Update
2768          Tune1();              // Conditional mtr1 PID tuning output
2769      }
2770
2771      ////////////////////////////////////
2772      //    V S P 2    S E R V I C E S     //
2773      ////////////////////////////////////
2774      if(TIMESVC & Vsp2Msk)    // If time to sample Motor Velocity
2775      {
2776          UpdateVelocity2();  // Update Motor Velocity
2777          Trajectory2();        // Conditional mtr2 trajectory
2778          PID2();               // Conditional mtr2 PID Control Update
2779          Tune2();              // Conditional mtr2 PID tuning output
2780      }
2781
2782
2783      ////////////////////////////////////////////////////////////////
2784      //       N E W   M O T O R   C M D   S E R V I C E S          //
2785      //                                                            //
2786      // 1) SerialCmdDispatch() starts any RS232 or I2C2 cmds.      //
2787      //    ReturnData() returns data and/or status for serial cmds.//
2788      //                                                            //
2789      // 2) CheckCmdTimeout() conditionally monitors Serial Cmd     //
2790      //    Interface and shuts down motors after a timeout.        //
2791      //                                                            //
2792      // 2) PotOrRc issues motor cmds depending on mode and TIMESVC.//
2793      //    If Mode is RCMIX or POTMIX, PotOrRc(0x01) handles Mtr2  //
2794      //    also.  If OverCurrent -NOW-, PotOrRc() is skipped.      //
2795      ////////////////////////////////////////////////////////////////
2796      SerialCmdDispatch();      // Handle any new cmd from serial interfaces
2797      ReturnData();             // Returns status or data from commands
2798      if(TIMESVC & CmdspMsk)  CheckCmdTimeout();  // Monitor serial cmd interface
2799      if(!(Mtr1_Flags1 & OverCurrent_Msk))     PotOrRc(0x01);  // Mtr1 POT or RC
2800      if(!(Mtr2_Flags1 & OverCurrent_Msk))     PotOrRc(0x02);  // Mtr2 POT or RC
2801  }
2802  //-------------------------------------------------------------------------
2803  void Svc1(void)           // INT0: AB2INT (Not currently used)
2804  {
2805  }
2806  //-------------------------------------------------------------------------
2807  void Svc2(void)           // INT1: AB1INT (Not currently used)
2808  {
2809  }
2810  //-------------------------------------------------------------------------
2811  void Svc3(void)           // TMR1: RTC (every 1000 msec)
2812  {
2813      ////////////////////////////////////////////////////////////////////
2814      // Note: Good place for services that don't require timely execution  //
2815      ////////////////////////////////////////////////////////////////////
2816      WORD x,vbatt;
2817                        // Computed VBATT voltage in mV.
2818
2819      ////////////////////////////////////////////////////////////////////
2820      //                        Monitor VBATT                           //
2821      //                                                                //
2822      //  Hysteresis avoids LED blinking near the warning threshold.    //
2823      ////////////////////////////////////////////////////////////////////
```

```
2824      x=AdcConvert(ADC0[6]);                    // V6 in millivolts
2825      vbatt=AdcToVbatt(x);                      // VBATT in millivolts
2826      if (vbatt < VBWARN)
2827      {
2828          LedErr |= VBATTmsk;                   // Signal Low VBATT
2829      }
2830      else if ( vbatt > (VBWARN + LED_HYSTERESIS) )
2831      {
2832          LedErr &= ~VBATTmsk;                  // Signal VBATT ok
2833      }
2834  }
2835  //-------------------------------------------------------------------------
2836  void Svc4(void)          // TMR2: ADC State Machine
2837  {
2838  }
2839  //-------------------------------------------------------------------------
2840  void Svc5(void)          // INT2: I2INT - Mtr2 Current Sense
2841  {
2842      ////////////////////////////////////////////////////////////////////
2843      //  The corresponding interrupt has occurred because the motor    //
2844      //  current sense voltage has transitioned into or out of voltage //
2845      //  threshold window specified by the on board digital pots.      //
2846      ////////////////////////////////////////////////////////////////////
2847      Motor2Current();     // Mtr2 Over Current Transition
2848  }
2849  //-------------------------------------------------------------------------
2850  void Svc6(void)          // INT3: I1INT - Mtr1 Current Sense
2851  {
2852      ////////////////////////////////////////////////////////////////////
2853      //  The corresponding interrupt has occurred because the motor    //
2854      //  current sense voltage has transitioned into or out of voltage //
2855      //  threshold window specified by the on board digital pots.      //
2856      ////////////////////////////////////////////////////////////////////
2857      Motor1Current();     // Mtr1 Over Current Transition
2858  }
2859  //-------------------------------------------------------------------------
2860  void Svc7(void)          // UNUSED
2861  {
2862  }
2863  //-------------------------------------------------------------------------
2864  void Svc8(void)          // TX1 (UNUSED)
2865  {
2866  }
2867  //-------------------------------------------------------------------------
2868  void Svc9(void)          // RC1 Service
2869  {
2870      BYTE err;
2871      // The RX1 ISR has indicated that a unit of data (either a null terminated
2872      // (ASCIZ) command string in the case of a Terminal Emulator App, or a
2873      // packetized command string in the case of a Smart App), is in the
2874      // circular buffer Rx1_Buff.  The code here parses the command string and
2875      // then if no errors causes it to be serviced later.
2876      //
2877      // TERMINAL EMULATOR CASE (See Owner's Manual for documentation)
2878      //    Eg:  The 11 char ASCIZ string  "L 01 01 3F"0x00 is parsed as:
2879      //
2880      //         CMD: "L"
2881      //         ARG[]: 0x01,0x01,0x3F
2882      //         ARGN: 3
2883      //
2884      // SMART APPLICATION CASE (See API Specification for documentation)
2885      //    Eg:  The xx char HEX string "02 01 4C 04 01 01 3F 20 49 03" is parsed as:
2886      //
2887      //         STX (0x02)
2888      //         NID (0x01)
2889      //           CMD: "L" (0x4C)
2890      //           N: ARGN=4 (0x04)
2891      //           ARG[]: 0x01,0x01,0x3F,0x20
2892      //           CKSUM: (0x49)
2893      //         ETX (0x03)
2894      //
2895      //         Note: API version of CMD_L has additional 4th argument (Length)
2896      //-------------------------
```

```
2897      if ( SCFG == APIcfg )
2898      {// Smart Application Uses API
2899          err = API_CmdParse();
2900          if (!err) { serialcmd=TRUE; CmdSource=API_SrcMsk; }
2901      } // If API
2902      //-----------------------------
2903      else if ( SCFG == TEcfg )
2904      {// Terminal Emulator Application uses ASCII string data
2905          err = TE_CmdParseExt();
2906          if (!err) { serialcmd=TRUE; CmdSource=TE_SrcMsk; }
2907          else if (err==eParseErr)     printf("Parse Err\r\nEnter 'H' for help\r\n");
2908      } // If TE
2909      //-----------------------------
2910  }
2911  //-------------------------------------------------------------------------
2912  void SvcA(void)          // CCP1 (EX0 - R/C Channel 1)
2913  {
2914      WORD pw;
2915
2916      ////////////////////////////////////////////////////////////////
2917      // Remark: The 16-bit PulseWidth is captured within the CCP ISR.  //
2918      // The ISR is briefly disabled here to avoid potential glitch in  //
2919      // obtaining the current value.                                   //
2920      ////////////////////////////////////////////////////////////////
2921      PIE1bits.CCP1IE = 0;         // Disable CCP1: Pulse capture
2922      pw = PulseWidth1;            // pw <= 0x7FFF
2923      PIE1bits.CCP1IE = 1;         // Enable CCP1:  Pulse capture
2924
2925      NewPulse |= NewRC1msk;       // Flag new pulse arrival
2926      RC[0] = PulseConvert(pw);
2927  }
2928  //-------------------------------------------------------------------------
2929  void SvcB(void)          // CCP2 (EX1 - R/C Channel 2)
2930  {
2931      WORD pw;
2932
2933      ////////////////////////////////////////////////////////////////
2934      // Remark: The 16-bit PulseWidth is captured within the CCP ISR.  //
2935      // The ISR is briefly disabled here to avoid potential glitch in  //
2936      // obtaining the current value.                                   //
2937      ////////////////////////////////////////////////////////////////
2938      PIE2bits.CCP2IE = 0;         // Disable CCP2: Pulse capture
2939      pw = PulseWidth2;            // pw <= 0x7FFF
2940      PIE2bits.CCP2IE = 1;         // Enable CCP2:  Pulse capture
2941
2942      NewPulse |= NewRC2msk;       // Flag new pulse arrival
2943      RC[1] = PulseConvert(pw);
2944  }
2945  //-------------------------------------------------------------------------
2946  void SvcC(void)          // CCP3 (EX3 - R/C Channel 3)
2947  {
2948      WORD pw;
2949
2950      ////////////////////////////////////////////////////////////////
2951      // Remark: The 16-bit PulseWidth is captured within the CCP ISR.  //
2952      // The ISR is briefly disabled here to avoid potential glitch in  //
2953      // obtaining the current value.                                   //
2954      ////////////////////////////////////////////////////////////////
2955      PIE3bits.CCP3IE = 0;         // Disable CCP3: Pulse capture
2956      pw = PulseWidth3;            // pw <= 0x7FFF
2957      PIE3bits.CCP3IE = 1;         // Enable CCP3:  Pulse capture
2958
2959      NewPulse |= NewRC3msk;       // Flag new pulse arrival
2960      RC[2] = PulseConvert(pw);
2961  }
2962  //-------------------------------------------------------------------------
2963  void SvcD(void)          // SSP2 (I2C2 SLAVE Serial Interface)
2964  {
2965      BYTE err;
2966
2967      // Application using I2C2 to issue commands
2968      err = I2C2_CmdParse();
2969      if (!err) { serialcmd=TRUE; CmdSource = I2C2_SrcMsk; }
```

```
2970 }
2971 //---------------------------------------------------------------------------
2972 void SvcE(void)          // UNUSED
2973 {
2974 }
2975 //---------------------------------------------------------------------------
2976 void SvcF(void)          // UNUSED
2977 {
2978 }
2979 //---------------------------------------------------------------------------
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990 //---------------------------------------------------------------------------
2991 void main (void)
2992 {
2993     SystemInitExt();             // Basic System Initialization
2994 //*************************************************************
2995 //**  SystemInit() is required.  If you need to reconfigure  **
2996 //**  resources, do it after SystemInit()                    **
2997 //*************************************************************
2998     #ifdef DALF_ROVIM_T2D
2999     ROVIM_T2D_Init();        //ROVIM System Initialization
3000
3001     /*The actions performed by this function do not need interrupts to work. However, print
            outputs
3002     are done through interrupts. The function seems to work fine at this stage of execution,
3003     so it will stay here*/
3004     ROVIM_T2D_Lockdown();  // Lock the brakes as soon as possible - safety first/
3005     #endif //DALF_ROVIM_T2D
3006
3007 #ifdef DALF_TEST_ENABLED
3008     TEST_TestInit();         //Testing Module Initialization
3009 #endif
3010
3011 #ifdef LOG_ENABLED
3012     LOG_LogInit();           //Internal non-volatile event logger initialization
3013 #endif
3014
3015     InitLED();               // ServiceLED Initialization
3016
3017     // Enable Interrupt System
3018     EnableInterrupts();
3019
3020 #ifdef WATCHDOG_ENABLED
3021     /*I don't think there's much problem in starting watchdog here instead of from power up,
3022     because until here the signals that are activated are goo (safetywise), not bad
3023     (such as acc, activate traction, etc.)*/
3024     STATUS_MSG("Starting watchdog.\r\n");
3025     DEBUG_MSG("Beware watchdog may actuate with debug traces enabled.\r\n");
3026     InitWatchdog();
3027 #endif //WATCHDOG_ENABLED
3028
3029     // Select custom putc()
3030     stdout = _H_USER;
3031     stderr = _H_USER;
3032
3033     WinkLEDS();              // Wink LED's to indicate power on reset.
3034
3035     #ifdef DALF_ROVIM_T2D
3036     ROVIM_T2D_Greeting();
3037     #endif //DALF_ROVIM_T2D
3038
3039     #ifdef DALF_ROVIM_T2D
3040     ROVIM_T2D_Start();
3041     #endif //DALF_ROVIM_T2D
```

```
3042
3043      //continuously monitor the changes we're doing, to avoid bigger troubles in the
3044      //future
3045  #ifdef DALF_TEST_ENABLED
3046      //TEST_InDevelopmentTesting();
3047      //TEST_StartGPIODriverTest();
3048  #endif
3049
3050  //  ***************************************************************************
3051  //  *                          M A I N    L O O P                             *
3052  //  *                                                                         *
3053  //  *  MAIN LOOP - Process service requests from ISR's (int service routines) *
3054  //  ***************************************************************************
3055      while(1)
3056      {
3057          SERVICE = SERVICE_REQ;              // SERVICE = Working copy
3058          SERVICE_REQ ^= SERVICE;             // Clear Svc Req Flags
3059          if( SERVICE )
3060          {
3061              if( SERVICE & 0x0001 ) Svc0();  // Mtr Cmd Processing
3062              if( SERVICE & 0x0008 ) Svc3();  // RTC
3063              if( SERVICE & 0x0020 ) Svc5();  // Mtr2 Current Sense
3064              if( SERVICE & 0x0040 ) Svc6();  // Mtr1 Current Sense
3065              if( SERVICE & 0x0200 ) Svc9();  // RC1
3066              if( SERVICE & 0x0400 ) SvcA();  // EX0 R/C Channel 1
3067              if( SERVICE & 0x0800 ) SvcB();  // EX1 R/C Channel 2
3068              if( SERVICE & 0x1000 ) SvcC();  // EX3 R/C Channel 3
3069              if( SERVICE & 0x2000 ) SvcD();  // SSP2 (I2C2)
3070
3071  //////////////////////////////////////////////////////////////////////////
3072  //  Unused  - Bit set by ISR, but no required main loop service         //
3073  //  Unmapped - Bit Doesn't get set by anything                          //
3074  //////////////////////////////////////////////////////////////////////////
3075          //  if( SERVICE & 0x0002 ) Svc1();  // Mtr2 Encoder    (UNUSED)
3076          //  if( SERVICE & 0x0004 ) Svc2();  // Mtr1 Encoder    (UNUSED)
3077          //  if( SERVICE & 0x0010 ) Svc4();  // TMR2: ADC State (UNUSED)
3078          //  if( SERVICE & 0x0080 ) Svc7();  // ----------------- (UNMAPPED)
3079          //  if( SERVICE & 0x0100 ) Svc8();  // TX1               (UNUSED)
3080          //  if( SERVICE & 0x4000 ) SvcE();  // ----------------- (UNMAPPED)
3081          //  if( SERVICE & 0x8000 ) SvcF();  // ----------------- (UNMAPPED)
3082          }
3083          // If enabled, and nothing to do, enter low power (IDLE) mode.
3084          if( (SYSMODE & IdleMsk) && (!SERVICE_REQ) ) DoIdle();
3085      }   // while(1)
3086  } // main()
3087  //-------------------------------------------------------------------------
3088  // High priority interrupt handler
3089  // #pragma interrupt ISRHI
3090  #pragma interrupt ISRHI nosave=section("MATH_DATA"),section(".tmpdata")
3091  void ISRHI (void)
3092  {
3093      // If INT2 Interrupt (I2INT: Mtr2 Current Sense)
3094      if ( (INTCON3bits.INT2IE == 1) && (INTCON3bits.INT2IF == 1) ) INT2_ISR();
3095
3096      // If INT3 Interrupt (I1INT: Mtr1 Current Sense)
3097      if ( (INTCON3bits.INT3IE == 1) && (INTCON3bits.INT3IF == 1) ) INT3_ISR();
3098
3099      // If TMR0 Interrupt (Heartbeat: 1msec)
3100      if ( (INTCONbits.TMR0IE == 1) && (INTCONbits.TMR0IF == 1) ) TMR0_ISR();
3101
3102      // If TMR1 Interrupt (RTC: 1sec)
3103      if ( (PIE1bits.TMR1IE == 1) && (PIR1bits.TMR1IF == 1) ) TMR1_ISR();
3104
3105      // If TMR2 Interrupt (ADC)
3106      if ( (PIE1bits.TMR2IE == 1) && (PIR1bits.TMR2IF == 1) ) TMR2_ISR();
3107
3108      // If INT0 Interrupt (AB2INT: Mtr2 encoder)
3109      if ( (INTCONbits.INT0IE == 1) && (INTCONbits.INT0IF == 1) ) INT0_ISR();
3110
3111      // If INT1 Interrupt (AB1INT: Mtr1 encoder)
3112      #ifdef DALF_ROVIM_T2D
3113      if ( (INTCON3bits.INT1IE == 1) && (INTCON3bits.INT1IF == 1) ) INT1_ISR_SINGLE_SOURCE();
3114      #else //DALF_ROVIM_T2D
```

```
3115      if ( (INTCON3bits.INT1IE == 1) && (INTCON3bits.INT1IF == 1) ) INT1_ISR();
3116      #endif //DALF_ROVIM_T2D
3117
3118      // If TX1 Interrupt (Cmd Interface Xmit)
3119      if ( (PIE1bits.TX1IE == 1) && (PIR1bits.TX1IF == 1) ) TX1_ISR();
3120
3121      // If RX1 Interrupt (Cmd Interface Rcv)
3122      if ( (PIE1bits.RC1IE == 1) && (PIR1bits.RC1IF == 1) ) RX1_ISR();
3123
3124      // If CCP1 Interrupt (EX0 - R/C Channel 1 [Mtr1])
3125      if ( (PIE1bits.CCP1IE == 1) && (PIR1bits.CCP1IF == 1) ) CCP1_ISR();
3126
3127      // If CCP2 Interrupt (EX1 - R/C Channel 2 [Mtr2])
3128      if ( (PIE2bits.CCP2IE == 1) && (PIR2bits.CCP2IF == 1) ) CCP2_ISR();
3129
3130      // If CCP3 Interrupt (EX2 - R/C Channel 3 [Alt])
3131      if ( (PIE3bits.CCP3IE == 1) && (PIR3bits.CCP3IF == 1) ) CCP3_ISR();
3132
3133      // If SSP2 Interrupt (SSP2 - Secondary I2C bus)
3134      if ( (PIE3bits.SSP2IE == 1) && (PIR3bits.SSP2IF == 1) ) SSP2_ISR();
3135  }
3136  //-----------------------------------------------------------------------------
3137  // Low priority interrupt handler
3138  #pragma interruptlow ISRHI
3139  void ISRLO (void)
3140  {
3141  }
3142
3143  #ifdef DALF_ROVIM_T2D
3144  // Custom handler for INT1 Interrupt (AB1INT: Mtr1 encoder), for a single-source encoder
3145  /* This ISR reads a single source encoder signal, and ignore quadrature signals and
3146  direction calculations.*/
3147  void INT1_ISR_SINGLE_SOURCE (void)
3148  {
3149      volatile BYTE A=0;
3150      static BYTE prevA=0;
3151
3152      INTCON3bits.INT1IF=0;
3153      /*Q: Should the counter be declared as volatile?
3154        A: I think it can work like this, because the variable may not be optimized in it's
               source
3155        object, and will therefore behave implicitly as volatile. But this is just an
               assumption.
3156        Anyhow, I've done some light testing, and it seems qualifying the variable as volatile
               doesn't
3157        change things.
3158        */
3159      /*change the polarity of the interrupt. Since we don't have a quadrature signal and we
             are
3160      getting a LOT of noisy interrupts (although the input signal doesn't appear to be noisy)
             ,
3161      the readings get corrupted if we just increment the counter on each interrupt (way more
3162      interrupts than encoder pulses). However, if we set the interrupt only on the rising
             edge, and
3163      compare the port value with the previous one, we only count the first interrupt, because
               after
3164      that the port value will allways equal the previous one. So we need to trigger the
             interrupt
3165      on both edges. To do that, we need to change the edge on each interrupt*/
3166      INTCON2bits.INTEDG1 = ~(INTCON2bits.INTEDG1);
3167      A=PORTE;
3168      A&=0x04; //We only want the RE2 bit, wich corresponds to the A1 input
3169      A=A>>2;
3170      if((A==1) && (prevA==0))
3171      {
3172          encode1++;
3173      }
3174      prevA=A;
3175  }
3176  #endif //DALF_ROVIM_T2D
3177  //-----------------------------------------------------------------------------
```

```
#line 1 "dalf_ext.c"          //work around the __FILE__ screwup on windows, http://www.microchip.com/forums/m746272.aspx
//cannot set breakpoints if this directive is used:
//info: http://www.microchip.com/forums/m105540-print.aspx
//uncomment only when breakpoints are no longer needed
/*********************************************************************************
**********************************************************************************
**
**
**                  dalf_ext.c - Generic software extensions and extended
**                               hardware support for the original Dalf
**                               firmware.
**
**       This module extends the original firmware of the Dalf-1F motor
**       control board to support generic hardware and software features
**       not included in the original version, that are not exclusive to
**       the T2D module.
**
**       This code was designed originally for the Dalf-1F motor control
**       board, the brain of the T2D module.
**       Original Dalf-1F firmware revision was 1.73.
**       See Dalf-1F owner's manual and the ROVIM T2D documentation for more
**       details.
**
**           The ROVIM Project
**********************************************************************************
**********************************************************************************/

#include "dalf.h"
#include "rovim.h"
#include <string.h>
#include "p18f6722.h"

static BOOL GetGPIOConfigbyName(const rom char* name, IOPinConfig* config);

static BYTE verbosity = VERBOSITY_DISABLED;      //controls the verbosity of the debug information

void (*AckCallback)(void) = NULL;

static BOOL ResourcesLockFlag=FALSE;

static const BYTE StandardCommandsToAllow[]={'C','E','G','H','I','K','L','O','P','R','U','V','X'};
static const BYTE nStandardCommandsToAllow=(BYTE) (sizeof(StandardCommandsToAllow)/sizeof(StandardCommandsToAllow[0]));
//Dalf extended commands ("G #") to block while on lockdown
static const BYTE ExtendedCommandsToAllow[]={0,1,2};
static const BYTE nExtendedCommandsToAllow=(BYTE) (sizeof(ExtendedCommandsToAllow)/sizeof(ExtendedCommandsToAllow[0]));

//////////////////////////////////////////////////////////////////////////////
//Debug reporting features.
//////////////////////////////////////////////////////////////////////////////

void DEBUG_PrintCmd(void)
{
    BYTE i;

    if ((!(verbosity & VERBOSITY_LEVEL_DEBUG)) || (SCFG != TEcfg))
        return;

    DEBUG_MSG("Cmd received: %c,(%d in decimal). # arguments: %d. Arguments: ", CMD, CMD, ARGN);
    for(i=0;i<ARGN;i++)
    {
        printf("%02X, ",ARG[i]);
    }
    printf("\r\n");
}

void SetVerbosity(BYTE level)
{
```

```
68      verbosity = VERBOSITY_DISABLED;
69
70      if (VERBOSITY_LEVEL_ERROR & level)
71          verbosity |= VERBOSITY_LEVEL_ERROR;
72      if (VERBOSITY_LEVEL_WARNING & level)
73          verbosity |= VERBOSITY_LEVEL_WARNING;
74      if (VERBOSITY_LEVEL_STATUS & level)
75          verbosity |= VERBOSITY_LEVEL_STATUS;
76      if (VERBOSITY_LEVEL_DEBUG & level)
77          verbosity |= VERBOSITY_LEVEL_DEBUG;
78      if (VERBOSITY_USE_CALL_INFO & level)
79          verbosity |= VERBOSITY_USE_CALL_INFO;
80      if (VERBOSITY_USE_TIMESTAMP & level)
81          verbosity |= VERBOSITY_USE_TIMESTAMP;
82  }
83
84  BYTE GetVerbosity(void)
85  {
86      return verbosity;
87  }
88
89  /*
90  To convert ticks to ms we divide by 32 instead of 33, to speed the calculation.
91  This creates an error that maxes at 24ms, or 2.4%. This is an acceptable trade-off
92  */
93  DWORD CalculateDelayMs(PTIME start, PTIME end)
94  {
95      ULONG i=0,j=0;
96      if ((start==NULL) || (end==NULL))
97      {
98          return -1;
99      }
100     i=TIME_TO_MSEC((*end));
101     j=TIME_TO_MSEC((*start));
102     if (j > i)
103     {
104         return -1;
105     }
106     return (DWORD)(i-j);
107 }
108
109
110 ////////////////////////////////////////////////////////////////////////////
111 //Logging features. This module is only compiled if the features are enabled.
112 ////////////////////////////////////////////////////////////////////////////
113
114
115 #ifdef LOG_ENABLED
116
117 void LOG_LogInit(void)
118 {
119     ERROR_MSG("LOG_LogInit not implemented.\r\n");
120     return;
121 }
122
123 #endif
124
125
126 ////////////////////////////////////////////////////////////////////////////
127 //system functions                                                        //
128 ////////////////////////////////////////////////////////////////////////////
129
130 void SystemInitExt(void)
131 {
132     //This has to be the first action of this function!!
133     SystemInit();
134     //TODO: find out if we just recovered from a hard reset (watchdog) or it's a power-on
135     return;
136 }
137
138 #ifdef WATCHDOG_ENABLED
139
140 void InitWatchdog(void)
```

```
{
    /*We enable only watchdog after system initialization, instead of from power on (On the
    config register), because when running DEBUG traces, the bootup time will be longer than
        the
    watchdog timeout, so the system will be constantly rebooting.
    Also, this way I can test this feature using the debugger - much faster and
        straightforward
    See PIC18F6722 datasheet, Section 25.2 - Watchdog timer for details.*/
    volatile BYTE *wdtcon;
    wdtcon = (volatile BYTE *)0xFD1;
    ((*(volatile BYTE *)wdtcon) = (0x01));

}

void KickWatchdog(void)
{
    ClrWdt();
    return;
}

void HardReset(void)
{
    Reset();
}

#endif

/**
\Brief Call the actions required to execute a command
*/
BYTE TeCmdDispatchExt(void)
{
    DEBUG_PrintCmd();
    if( IsStandardCommandLocked(CMD) )
    {
        return eDisable;
    }
    switch(CMD)
    {
        case 'G':
            if (ARGN==0)
            {
                return TeProcessAck();
            }
            if( IsExtendedCommandLocked(ARG[0]) )
            {
                return eDisable;
            }
            switch (ARG[0])
            {
                /*custom functions index:
                this is to be used if we want other custom functions
                other than ROVIM_T2D ones. They must be inserted here,
                to be parsed before calling the ROVIM dispatch*/
                case 0:
                    return TeProcessAck();
                case 1:
                    return TeDisableAck();
                default:
                    #ifdef DALF_ROVIM_T2D
                    return ROVIM_T2D_CmdDispatch();
                    #else //DALF_ROVIM_T2D
                    return eParseErr
                    #endif //DALF_ROVIM_T2D
            }
            break;
        case 'H':
        #ifdef HELP_ENABLED
            return ShowHelp();
        #else
            return eDisable;
        #endif
            break;
```

```
212         case 'O':
213             #ifdef DALF_ROVIM_T2D
214             WARNING_MSG("Motor configurations lost. Make sure to restore them before \
215 normal operation. You may have to reboot.\r\nYou should use the ROVIM T2D stop motors
        instead.\r\n");
216             #endif //DALF_ROVIM_T2D
217             //fall through
218         default:
219             return TeCmdDispatch();
220             break;
221     }
222     return eParseErr;
223 }
224
225 /**
226 \Brief Parse the characters from the uart buffer into variables describing a command
227 */
228 BYTE TE_CmdParseExt(void)
229 {
230     return TE_CmdParse();
231 }
232
233 BYTE I2C2CmdDispatchExt(void)
234 {
235     return I2C2CmdDispatch();
236 }
237
238 BYTE TeProcessAck(void)
239 {
240     if(ARGN > 1)
241     {
242         return eNumArgsErr;
243     }
244
245     if(AckCallback != NULL)
246     {
247         AckCallback();
248     }
249     //call registered functions to process the ack
250     STATUS_MSG("There are no tasks pending user validation to resume.\r\n");
251     return NoErr;
252 }
253
254 BYTE TeDisableAck(void)
255 {
256     if(ARGN != 1)
257     {
258         return eNumArgsErr;
259     }
260
261     AckCallback=NULL;
262     STATUS_MSG("Tasks pending user validation were terminated.\r\n");
263 }
264
265 #ifdef HELP_ENABLED
266 // Help submodule
    ------------------------------------------------------------------------------
267 BYTE ShowHelp(void)
268 {
269     BYTE i=0;
270     #ifdef DALF_ROVIM_T2D
271     //temporary, quick help
272     MSG("Dalf ROVIM T2D Help.\r\nUsage: H\r\nAvailable help topics:\r\n");
273     MSG("\tC\t\t\tRead ADC.\r\n\tE\t\t\tShow encoder count.\r\n\tG[cmd code] [args...]\
            tCustom \
274 Commands. See below.\r\n\tH\t\t\tShow this message.\r\n\tI\t\t\tReset software.\r\n\tU\t\t\
        tShow motor\
275  status.\r\n\tV\t\t\tShow motors velocity.\r\n\tX2 [dir] [duty cycle]\tMove direction motor
         in open \
276 loop.\r\n[dir] can be: '0' - move to port, or '1' - move to starboard.\r\n\nCustom commands
        :\r\n");
277     MSG("\tG10\t\t\tGo to lockdown.\r\n");
278     MSG("\tG11\t\t\tRelease from lockdown, if the system is good to go.\r\n");
```

```
279     MSG("\tG12 [mtr]\t\tStop motor [mtr]. If [mtr] is omitted, both motors are stopped.\r\n"
            );
280     MSG("\tG14 PWM\t\t\tSet the decelerator (braking energy recovery) to [PWM]%%.\r\n");
281     MSG("\tG15 PWM\t\t\tSet the accelerator to [PWM]%%.\r\n");
282     MSG("\tG16 type speed\t\tSet the vehicle to move in direction [type] at speed [speed].\r
            \n\
283 Possible directions are:\r\n0 - Forward;\r\n1 - Reverse;\r\n2 - Neutral (coast);\r\n3 - Hill
        hold (maintain position).\r\n\
284 Speed is expressed in tenths of Km/h (km/10/h) and varies between %d and %d.\r\n",0,
        ROVIM_T2D_MAX_SPEED);
285     MSG("\tG17 angle\t\tturn direction to [angle].\r\n\n");
286     MSG("Do not forget that all inputs through this interface are in hexadecimal and all
            outputs of \
287 the standard commands are also in hexadecimal. All other messages and custom commands
        outputs are in decimal.\r\n");
288     MSG("Example usage:\r\nX2 00 30 - Moves the direction to port at 48%% of full power\r\n\
289 G16 00 20 - Moves the vehicle forward at ~3.2 Km/h\r\n\G17 1B - Turn the vehicle 16Âº to
        starboard.\r\n\n");
290     MSG("GPIO pin count on Dalf board J5 and J6 expanders:\r\n(male socket seen from above)\
            r\n-----------------------\r\n|1 3 5 ....... 15|\
291 \r\n|2 4 6 ....... 16|\r\n-------- --------\r\nGPIOs are:\r\n");
292     for(i=0;i<ngpios;i++)
293     {
294         if(strcmppgm("",GPIOsDescription[i].name)!=0)
295         {
296             MSG("%d - %s\r\n",(i+1), GPIOsDescription[i].name);
297         }
298     }
299     //True beauty lies in ascii art...
300     MSG("\nDirection angle representation (motorcycle seen from above):\r\n            %dÂº\
            r\n\
301    %dÂº  \\---|---/ 0Âº\r\n          \\  |  /  \r\n           \\ | /   \r\n            \\|/\
           r\n         _ (FRONT) _\r\n\
302         |_|---|---|_|\r\n          |     |\r\n  (PORT)  |     |  (STARBOARD)\r\n          |
                  |\r\n\
303         _ |     | _\r\n       |_|-------|_|\r\n           (REAR)  \r\n\n\
304 The direction angle (the argument for command G17) starts at full starboard - 0Âº and goes
        until full port - %dÂº.\r\n",
305  (ROVIM_T2D_DIR_ANGULAR_RANGE/2), ROVIM_T2D_DIR_ANGULAR_RANGE, ROVIM_T2D_DIR_ANGULAR_RANGE);
306     MSG("Motor 1 is the traction motor and motor 2 is the direction motor. However, motor 1
            must be \
307 controlled through the custom commands - motor 2 can also be controlled by the standard
        commands.\r\n");
308     MSG("Dalf board led blinking information:\r\nLED1 ON - traction motor is either moving
            or ready to move through auto mode.\r\n\
309 LED1 OFF - traction motor is not ready to move with auto mode.\r\nLED2 ON or blinking -
        direction motor is either moving\
310  or being powered.\r\nLED2 OFF - direction motor is not being powered.\r\nLED3 OFF - No
         errors.\r\n\
311 LED3 blinking - ROVIM is in lockdown mode.\r\nMTR1 ON green - traction motor is being
        accelerated\
312  on auto mode.\r\nMTR1 ON red - traction motor is being decelerated on auto mode.\r\nMTR2 ON
          green - direction is moving to port.\r\n\
313 MTR2 ON red - direction is moving to starboard.\r\nMTR1 and MTR2 intensity depends on the
        PWM duty cycle.\r\n");
314     MSG("For further reference please consult the users manual.\r\n");
315     return NoErr;
316     #else
317     ERROR_MSG("ShowHelp not implemented.\r\n");
318     #endif
319     return eDisable;
320 }
321
322 void RegisterCmdhelp(void)
323 {
324     ERROR_MSG("RegisterCmdhelp not implemented.\r\n");
325     return;
326 }
327
328 #endif
329
330 void LockCriticalResourcesAccess(void)
331 {
```

```
332     /*given which and how GPIOs are currently used, they do not need special protection. If
            needed
333     in the future, it should be added.*/
334     ResourcesLockFlag=TRUE;
335     #ifdef DALF_ROVIM_T2D
336     ROVIM_T2D_LockCriticalResourcesAccess();
337     #endif
338     return;
339 }
340
341 void UnlockCriticalResourcesAccess(void)
342 {
343     ResourcesLockFlag=FALSE;
344     #ifdef DALF_ROVIM_T2D
345     ROVIM_T2D_UnlockCriticalResourcesAccess();
346     #endif
347     return;
348 }
349
350 BOOL IsStandardCommandLocked(BYTE cmd)
351 {
352     BYTE i;
353
354     if(!ResourcesLockFlag)
355     {
356         return FALSE;
357     }
358
359     for(i=0;i<nStandardCommandsToAllow;i++)
360     {
361         if( cmd==StandardCommandsToAllow[i])
362         {
363             DEBUG_MSG("Found cmd=%d is in whitelist position %d.\r\n", cmd, i);
364             return FALSE;
365         }
366     }
367     ERROR_MSG("Command is locked.\r\n");
368     return TRUE;
369 }
370
371 BOOL IsExtendedCommandLocked(BYTE cmd)
372 {
373     BYTE i;
374
375     if(!ResourcesLockFlag)
376     {
377         return FALSE;
378     }
379     if (cmd >= CUSTOM_CMD_ID_OFFSET)
380     {
381         //command does not exist in extended command set. nExtendedCommandsToLock[] is
                incorrect
382         DEBUG_MSG("Command does not belong to dalf extended set.\r\n");
383         return FALSE;
384     }
385
386     for(i=0;i<nExtendedCommandsToAllow;i++)
387     {
388         if( cmd==ExtendedCommandsToAllow[i])
389         {
390             DEBUG_MSG("Found cmd=%d is in whitelist position %d.\r\n", cmd, i);
391             return FALSE;
392         }
393     }
394     DEBUG_MSG("Command is locked.\r\n");
395     return TRUE;
396 }
397
398 // GPIO sub-driver
    ------------------------------------------------------------------------------
399 /*Remark: Due to the nature of this application, where all possible GPIOs are known
        beforehand,
```

```
400 (since they need to be physically connected) this driver works with a constant set of
        possible
401 GPIOs, while allowing to change each one's configuration. Each GPIO has a default
        configuration,
402 that is the mos likely to be use during program operation.*/
403
404 //Set a config for an existing GPIO
405 BOOL SetGPIOConfig(const rom char* name, IOPinConfig* config)
406 {
407     BYTE aux=0, i=0;
408     BYTE dir=0;
409     BYTE pullup=0;
410     BYTE inverted=0;
411     //These variables depecting the previous state exist to provide debug information
412     BYTE previousDir=0;
413     BYTE previousPullup=0;
414     BYTE previousInverted=0;
415     IOPinConfig defaultConfig={0};
416
417     //parameter check
418     if(config == NULL)
419     {
420         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
421         return FALSE;
422     }
423     if((config->number > GPIOS_PER_EXPANDER) || (config->number==0))
424     {
425         ERROR_MSG("GPIO description parameters invalid. Aborting operation.\r\n");
426         return FALSE;
427     }
428     if(name == NULL)
429     {
430         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
431         return FALSE;
432     }
433     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
434     {
435         if(name[i] == '\0') break;
436     }
437     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
438     {
439         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
440         return FALSE;
441     }
442     //Verify the GPIO exists and is coherent with default params
443     if(!GetGPIOConfigbyName(name, &defaultConfig))
444     {
445         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
446         return FALSE;
447     }
448     //XXX:this should be part of the GPIO id. The config should only the 3 non-constant
        parameters
449     if((defaultConfig.number != config->number) || (defaultConfig.exp != config->exp))
450     {
451         ERROR_MSG("GPIO description does not match a valid GPIO's. Aborting operation.\r\n")
            ;
452         return FALSE;
453     }
454
455     //Get current IOEXP bank config
456     if(config->exp == J5){
457         previousDir=ReadIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config->number));
458         previousInverted=ReadIOExp2(IPOLA + IOEXP_REG_BANK_OFFSET(config->number));
459         previousPullup=ReadIOExp2(GPPUA + IOEXP_REG_BANK_OFFSET(config->number));
460     }
461     else{
462         previousDir=ReadIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config->number));
463         previousInverted=ReadIOExp1(IPOLA + IOEXP_REG_BANK_OFFSET(config->number));
464         previousPullup=ReadIOExp1(GPPUA + IOEXP_REG_BANK_OFFSET(config->number));
465     }
466
467     //Change the configuration of only the GPIO under treatment
468     /* For further info on these calculations see:
```

```
469     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in
            -c-c
470     CHANGE Nth BIT TO X: number ^= (-x ^ number) & (1 << n);*/
471     aux= config->inverted==ON? 0xFF:0;
472     inverted = previousInverted ^( (aux ^ previousInverted) & (1U << IOEXP_PIN_BIT_OFFSET(
            config->number)) );
473     aux= config->pullup==ON? 0xFF:0;
474     pullup = previousPullup ^( (aux ^ previousPullup) & (1U << IOEXP_PIN_BIT_OFFSET(config->
            number)) );
475     aux= config->dir==IN? 0xFF:0;
476     dir = previousDir ^( (aux ^ previousDir) & (1U << IOEXP_PIN_BIT_OFFSET(config->number))
            );
477
478     //Write the new configuration
479     if(config->exp == J5){
480         WriteIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config->number), dir);
481         WriteIOExp2(IPOLA + IOEXP_REG_BANK_OFFSET(config->number), inverted);
482         WriteIOExp2(GPPUA + IOEXP_REG_BANK_OFFSET(config->number), pullup);
483     }
484     else{
485         WriteIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config->number), dir);
486         WriteIOExp1(IPOLA + IOEXP_REG_BANK_OFFSET(config->number), inverted);
487         WriteIOExp1(GPPUA + IOEXP_REG_BANK_OFFSET(config->number), pullup);
488     }
489
490     DEBUG_MSG("SetGPIOConfig: exp=J%d, pin=%d, bank=%c, bank bit=%d. Previous: dir=%08b, \
491 inv=%08b, pull=%08b. Desired: dir=%01b, inv=%01b, pull=%01b. Current: dir=%08b, inv=%08b,
        pull=%08b.\r\n", \
492   (config->exp==J5)?5:6, config->number, ('A'+IOEXP_REG_BANK_OFFSET(config->number)),\
493 (IOEXP_PIN_BIT_OFFSET(config->number)+1), previousDir, previousInverted, previousPullup, \
494 (config->dir==IN? 1:0), (config->inverted==ON? 1:0),(config->pullup==ON? 1:0), dir, inverted
        , pullup);
495
496     /* Uncomment this section if you want to check intermediate calculations
497     DEBUG_MSG("SetGPIOConfig: pin=%d, bank offset=%d, bit offset=%d. Previous: dir=%08b, inv
            =%08b, pull=%08b\
498 . Intermediate calculations (for dir only): aux=%08b, aux^previousDir=%08b, 1U<<offset=%08b,
          (aux^previousDir) & (1U<<offset)\
499 =%08b. Current: dir=%08b, inv=%08b, pull=%08b.\r\n", config->number, IOEXP_REG_BANK_OFFSET(
        config->number),\
500 IOEXP_PIN_BIT_OFFSET(config->number), previousDir, previousInverted, previousPullup, aux, (
        aux ^ previousDir),\
501  (1U << IOEXP_PIN_BIT_OFFSET(config->number)), \
502  ((aux ^ previousDir) & (1U << IOEXP_PIN_BIT_OFFSET(config->number))), dir, inverted, \
503  pullup);*/
504
505     return TRUE;
506 }
507
508 //both parameters must point to valid variables. Space won't be allocated here
509 BOOL GetGPIOConfig(const rom char* name, IOPinConfig* config)
510 {
511     BYTE dir=0, pullup=0, inverted=0, aux=0, i=0;
512
513     if(config == NULL){
514         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
515         return FALSE;
516     }
517     if(name == NULL){
518         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
519         return FALSE;
520     }
521     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
522     {
523         if(name[i] == '\0') break;
524     }
525     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
526     {
527         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
528         return FALSE;
529     }
530     if(!GetGPIOConfigbyName(name, config)){
531         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
```

```
532         config=NULL;
533         return FALSE;
534     }
535
536     //read the IOEXP bank configuration
537     if(config->exp == J5){
538         dir=ReadIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config->number));
539         inverted=ReadIOExp2(IPOLA + IOEXP_REG_BANK_OFFSET(config->number));
540         pullup=ReadIOExp2(GPPUA + IOEXP_REG_BANK_OFFSET(config->number));
541     }
542     else{
543         dir=ReadIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config->number));
544         inverted=ReadIOExp1(IPOLA + IOEXP_REG_BANK_OFFSET(config->number));
545         pullup=ReadIOExp1(GPPUA + IOEXP_REG_BANK_OFFSET(config->number));
546     }
547
548     //get and translate the configuration for the GPIO under treatment
549     /* For further info on these calculations see:
550     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in-c-c
551     CHECK: bit = (number >> x) & 1;*/
552     aux= (dir >> IOEXP_PIN_BIT_OFFSET(config->number)) & 1U;
553     config->dir= aux? IN: OUT;
554     aux= (inverted >> IOEXP_PIN_BIT_OFFSET(config->number)) & 1U;
555     config->inverted= aux? ON: OFF;
556     aux= (pullup >> IOEXP_PIN_BIT_OFFSET(config->number)) & 1U;
557     config->pullup= aux? ON: OFF;
558
559     DEBUG_MSG("GetGPIOConfig: exp=J%d, pin=%d, bank=%c, bank bit=%d. dir=%08b,%d, inv=%08b,%d, pull=%08b,\
560 %d.\r\n", (config->exp==J5)?5:6, config->number, ('A'+IOEXP_REG_BANK_OFFSET(config->number)),\
561     (IOEXP_PIN_BIT_OFFSET(config->number)+1), dir, config->dir, inverted, config->inverted,\
562     pullup, config->pullup);
563     return TRUE;
564 }
565
566 BOOL CompareGPIOConfig(IOPinConfig* config1, IOPinConfig* config2)
567 {
568     if(config1 == NULL){
569         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
570         return FALSE;
571     }
572     if(config2 == NULL){
573         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
574         return FALSE;
575     }
576     if((config1->number > GPIOS_PER_EXPANDER) || (config1->number==0))
577     {
578         ERROR_MSG("GPIO config parameters invalid. Aborting operation.\r\n");
579         return FALSE;
580     }
581     if((config2->number > GPIOS_PER_EXPANDER) || (config2->number==0))
582     {
583         ERROR_MSG("GPIO config parameters invalid. Aborting operation.\r\n");
584         return FALSE;
585     }
586
587     DEBUG_MSG("CompareGPIOConfig: 1:exp=J%d,number=%d,dir=%d,pullup=%d,inverted=%d;  2:exp=J%d,\
588 number=%d,dir=%d,pullup=%d,inverted=%d\r\n",(config1->exp==J5)?5:6, config1->number,config1->dir,\
589 config1->pullup,config1->inverted, (config2->exp==J5)?5:6, config2->number,config2->dir,
        config2->pullup,config2->inverted);
590
591     if( memcmp( (const void *)config1, (const void *)config2, sizeof(IOPinConfig)))
592     {
593         return FALSE; //configurations do not match
594     }
595
596     return TRUE;
597 }
598
```

```
599 //String constants are automatically stored in rom. See MPLAB C-18 Users guide, 2.73.
600 /*this function searches the list of Registered GPIO names (set during initialization) for a
601 match to the name provided. If it is found, it returns the GPIO's corresponding
        configuration,
602 If not, an error occurs.*/
603 //This is an sub-driver private function. Parameter check is done before calling it.
604 static BOOL GetGPIOConfigbyName(const rom char* name, IOPinConfig* config)
605 {
606     BYTE i=0;
607
608     if(config == NULL){
609         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
610         return FALSE;
611     }
612     if(name == NULL){
613         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
614         return FALSE;
615     }
616
617     for(i=0; i<ngpios; i++){
618         //Do NOT use strncmppgm, since it requires the 3rd argument to be in rom and fails
              silently if it isn't correctly provided!
619         if(strcmppgm(name, GPIOsDescription[i].name) == 0){
620             memcpypgm2ram(config,(const rom void *) &GPIOsDescription[i].config,sizeof(*
                  config));
621             //DEBUG_MSG("GetGPIOConfigbyName: matching name found for '%HS'.\r\n", name);
                  //%HS prints a string located in far rom
622             return TRUE;
623         }
624     }
625
626     ERROR_MSG("GetGPIOConfigbyName could not find a matching name for '%HS'.\r\n",name);
627     config=NULL;
628     return FALSE;
629 }
630
631 //"Rule of thumb: Always read inputs from PORTx and write outputs to LATx. If you need to
        read what you set an output to, read LATx."
632 BOOL SetGPIO(const rom char* name)
633 {
634     BYTE i=0;
635     BYTE previousValue=0, value;
636     BYTE dir=0;
637     IOPinConfig config;
638
639     if(name == NULL){
640         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
641         return FALSE;
642     }
643     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
644     {
645         if(name[i] == '\0') break;
646     }
647     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
648     {
649         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
650         return FALSE;
651     }
652     if(!GetGPIOConfigbyName(name, &config)){
653         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
654         return FALSE;
655     }
656     //Check if the GPIO is configured as input
657     if(config.exp == J5){
658         dir=ReadIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
659     }
660     else{
661         dir=ReadIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
662     }
663     dir = (dir >> IOEXP_PIN_BIT_OFFSET(config.number)) & 1U;
664     if(dir){
665         ERROR_MSG("GPIO configured as input. Aborting operation.\r\n");
666         return FALSE;
```

```
667     }
668
669     //Get current IOEXP bank gpios
670     if(config.exp == J5){
671         previousValue=ReadIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
672     }
673     else{
674         previousValue=ReadIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
675     }
676
677     //Change the configuration of only the GPIO under treatment
678     /* For further info on these calculations see:
679     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in
        -c-c
680     SET: number |= 1 << x;*/
681     value = previousValue | (1U << IOEXP_PIN_BIT_OFFSET(config.number));
682
683     //Write the new value only if needed
684     if(previousValue!=value)
685     {
686         if(config.exp == J5){
687             WriteIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
688         }
689         else{
690             WriteIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
691         }
692     }
693     else
694     {
695         /*DEBUG_MSG("GPIO value already matches pretended value. No action needed.\r\n");*/
696     }
697
698     DEBUG_MSG("SetGPIO: exp=J%d, pin=%d, bank %c, bank bit=%d. Previous bit value=%b,
        current \
699 bit value=%b.\r\n", (config.exp==J5)?5:6, config.number, ('A' + IOEXP_REG_BANK_OFFSET(config
    .number)),\
700     (IOEXP_PIN_BIT_OFFSET(config.number) +1), previousValue, value);
701     return TRUE;
702 }
703
704 BOOL ResetGPIO(const rom char* name)
705 {
706     BYTE i=0;
707     BYTE previousValue=0, value=0;
708     BYTE dir=0;
709     IOPinConfig config;
710
711     if(name == NULL){
712         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
713         return FALSE;
714     }
715     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
716     {
717         if(name[i] == '\0') break;
718     }
719     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
720     {
721         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
722         return FALSE;
723     }
724     if(!GetGPIOConfigbyName(name, &config)){
725         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
726         return FALSE;
727     }
728     //Check if the GPIO is configured as input
729     if(config.exp == J5){
730         dir=ReadIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
731     }
732     else{
733         dir=ReadIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
734     }
735     dir = (dir >> IOEXP_PIN_BIT_OFFSET(config.number)) & 1U;
736     if(dir){
```

```
737         ERROR_MSG("GPIO configured as input. Aborting operation.\r\n");
738         return FALSE;
739     }
740
741     //Get current IOEXP bank gpios
742     if(config.exp == J5){
743         previousValue=ReadIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
744     }
745     else{
746         previousValue=ReadIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
747     }
748
749     //Change the configuration of only the GPIO under treatment
750     /* For further info on these calculations see:
751     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in
            -c-c
752     RESET: number &= ~(1 << x);*/
753     value = previousValue & (~(1U << IOEXP_PIN_BIT_OFFSET(config.number)));
754
755     //Write the new value only if needed
756     if(previousValue!=value)
757     {
758         if(config.exp == J5){
759             WriteIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
760         }
761         else{
762             WriteIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
763         }
764     }
765     else
766     {
767         /*DEBUG_MSG("GPIO value already matches pretended value. No action needed.\r\n");*/
768     }
769
770     DEBUG_MSG("ResetGPIO: exp=J%d, pin=%d, bank %c, bank bit=%d. Previous bit value=%b,
            current \
771 bit value=%b.\r\n", (config.exp==J5)?5:6, config.number, ('A' + IOEXP_REG_BANK_OFFSET(config
        .number)),\
772     (IOEXP_PIN_BIT_OFFSET(config.number) +1), previousValue, value);
773     return TRUE;
774 }
775
776 BOOL ToggleGPIO(const rom char* name)
777 {
778     BYTE i=0;
779     BYTE previousValue=0, value=0;
780     BYTE dir=0;
781     IOPinConfig config;
782
783     if(name == NULL){
784         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
785         return FALSE;
786     }
787     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
788     {
789         if(name[i] == '\0') break;
790     }
791     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
792     {
793         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
794         return FALSE;
795     }
796     if(!GetGPIOConfigbyName(name, &config)){
797         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
798         return FALSE;
799     }
800     //Check if the GPIO is configured as input
801     if(config.exp == J5){
802         dir=ReadIOExp2(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
803     }
804     else{
805         dir=ReadIOExp1(IODIRA + IOEXP_REG_BANK_OFFSET(config.number));
806     }
```

```
807     dir = (dir >> IOEXP_PIN_BIT_OFFSET(config.number)) & 1U;
808     if(dir){
809         ERROR_MSG("GPIO configured as input. Aborting operation.\r\n");
810         return FALSE;
811     }
812 
813     //Get current IOEXP bank gpios
814     if(config.exp == J5){
815         previousValue=ReadIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
816     }
817     else{
818         previousValue=ReadIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number));
819     }
820 
821     /* For further info on these calculations see:
822     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in
        -c-c
823     TOGGLE: number ^= 1 << x;*/
824     value = previousValue ^ (1U << IOEXP_PIN_BIT_OFFSET(config.number));
825 
826     //Write the new value
827     if(config.exp == J5){
828         WriteIOExp2(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
829     }
830     else{
831         WriteIOExp1(OLATA + IOEXP_REG_BANK_OFFSET(config.number), value);
832     }
833 
834     DEBUG_MSG("ToggleGPIO: exp=J%d, pin=%d, bank %c, bank bit=%d. Previous bit value=%b,
        current \
835 bit value=%b.\r\n", (config.exp==J5)?5:6, config.number, ('A' + IOEXP_REG_BANK_OFFSET(config
    .number)),\
836     (IOEXP_PIN_BIT_OFFSET(config.number) +1), previousValue, value);
837     return TRUE;
838 }
839 
840 BOOL GetGPIO(const rom char* name, BYTE* value)
841 {
842     BYTE i=0;
843     IOPinConfig config;
844     BYTE aux=0;
845 
846     if(name == NULL){
847         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
848         return FALSE;
849     }
850     for(i=0;i<IO_PIN_NAME_MAX_LEN;i++)
851     {
852         if(name[i] == '\0') break;
853     }
854     if((i==0) || (i==IO_PIN_NAME_MAX_LEN))
855     {
856         ERROR_MSG("GPIO name invalid. Aborting operation.\r\n");
857         return FALSE;
858     }
859     if(!GetGPIOConfigbyName(name, &config)){
860         ERROR_MSG("GPIO does not exist. Aborting operation.\r\n");
861         return FALSE;
862     }
863     if(value == NULL){
864         ERROR_MSG("Received NULL input parameter. Aborting operation.\r\n");
865         return FALSE;
866     }
867 
868     if(config.exp == J5){
869         aux=ReadIOExp2(GPIOA + IOEXP_REG_BANK_OFFSET(config.number));
870     }
871     else{
872         aux=ReadIOExp1(GPIOA + IOEXP_REG_BANK_OFFSET(config.number));
873     }
874 
875     /* For further info on these calculations see:
```

```
876     http://stackoverflow.com/questions/47981/how-do-you-set-clear-and-toggle-a-single-bit-in-c-c
877     CHECK: bit = (number >> x) & 1;*/
878     *value= (aux>> IOEXP_PIN_BIT_OFFSET(config.number)) & 1U;
879
880     DEBUG_MSG("GetGPIO: exp=J%d, pin=%d, bank %c, bank bit=%d. Value=%d.\r\n", (config.exp==J5)?5:6,\
881     config.number, ('A' + IOEXP_REG_BANK_OFFSET(config.number)),\
882     (IOEXP_PIN_BIT_OFFSET(config.number) +1), *value);
883     return TRUE;
884 }
885
886 BOOL GetAllGPIO(BYTE* J5A, BYTE* J5B, BYTE* J6A, BYTE* J6B)
887 {
888     *J5A=ReadIOExp2(GPIOA);
889     *J6A=ReadIOExp1(GPIOA);
890     *J5B=ReadIOExp2(GPIOB);
891     *J6B=ReadIOExp1(GPIOB);
892
893     return TRUE;
894 }
```

```
1  /*****************************************************************************
2  *****************************************************************************
3  **
4  **
5  **                        dalf.h
6  **       This is the dalf.h file of the Dalf-1F v1.73 firmware, modified
7  **           for the ROVIM project.
8  **       See Dalf-1F owner's manual and the ROVIM T2D documentation for more
9  **           details.
10 **
11 **           The ROVIM Project
12 *****************************************************************************
13 *****************************************************************************/
14
15 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
16 //                                                                           x
17 //           D D          AA       L         FFFFFF      H    H            x
18 //           D   D       A   A     L         F           H    H            x
19 //           D    D      AAAAAA    L         FFFF        HHHHHH            x
20 //           D    D      A    A    L         F           H    H            x
21 //           D  D        A    A    LLLLLL    F      O    H    H            x
22 //                                                                           x
23 //                                                                           x
24 //    <dalf.h> -  Various definitions for the Dalf Motor Control Board       x
25 //                                                                           x
26 //  (c) Copyright 2006 Embedded Electronics LLC, All rights reserved         x
27 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
28
29
30 ////////////////////////////////////////////////////////////////////////////
31 //                          -- HISTORY --                                  //
32 //                                                                         //
33 //  --Date--   -Who-  ------------Description-------------------------     //
34 //  05/13/05    SMB    Created File                                        //
35 //                                                                         //
36 ////////////////////////////////////////////////////////////////////////////
37
38 #ifndef __DALF_H
39 #define __DALF_H
40
41 #include <stdio.h>
42
43
44 // Logic
45 #define FALSE   0x00
46 #define TRUE    ~FALSE              // The truth is not out there, it is here...
47
48 // ASCII Control Chrs
49 #define chr_STX     0x02            // Start of API Pkt[]
50 #define chr_ETX     0x03            // End of API Pkt[]
51 #define chr_OK      0xAA            // Successful Command Code
```

```
52
53 #define HOST_NID     0x00
54
55 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
56 //xx  C O N D I T I O N A L     F L A G S   xx
57 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
58 #define Dbg      0x00                // No debugging
59 //#define   Dbg      0x01                // Debug closed loop continuity
60
61
62 ////////////////////////////////////////////////////////////////////////////
63 //                        E R R O R     L I S T                           //
64 ////////////////////////////////////////////////////////////////////////////
65 #define NoErr        0           // Success
66 #define eErr         1           // Generic error
67 #define eParseErr    2           // Syntax (eg; unexpected buffer empty)
68 #define eNumArgsErr  3           // Arg count
69 #define eParmErr     4           // Bad parameter value
70 #define eModeErr     5           // Invalid operating mode
71 #define eFrameErr    6           // Framing Error
72 #define eOvRunErr    7           // OverRun Error
73 #define eBuffFull    8           // Buffer Full while receiving data.
74 #define eProtocol    9           // Packet Protocol Error.
75 #define eChkSum      10          // Check Sum.
76 #define eTimeOut     11          // Timeout.
77 #define eDisable     12          // Interface disabled.
78
79 //define absolute
80 #define abs(x) ((x) > 0 ? (x) : -(x))
81
82 ////////////////////////////////////////////////////////////
83 //                     S T R U C T U R E S                //
84 ////////////////////////////////////////////////////////////
85 typedef unsigned char   BYTE, UCHAR;        //  8 bits; [0 .. 255]
86 typedef unsigned int    WORD, UINT, USHORT; // 16 bits; [0 .. 65,535]
87 typedef unsigned long   DOUBLE, ULONG;      // 32 bits; [0 .. 4,294,967,295]
88 typedef long            LONG, DWORD;        // 32 bits signed;
89 typedef UCHAR BOOL;
90
91 /*  I/O expanders pin numbering information for Dalf board.
92     -IO expander 1 is connector J5
93     -IO expander 2 is connector J6
94     -Female I/O connector (attached to board) pin numbering, and translation to male
           connector
95     pin numbering:
96     ___________________________     \   ___________________________
97     | B1 B3 B5 B7 A6 A4 A2 A0 |   ===\  | 15 13 11 09 07 05 03 01 |
98     | B0 B2 B4 B6 A7 A5 A3 A1 |   ===/  | 16 14 12 10 08 06 04 02 |
99     ___________     ___________     /   ___________   ___________
100                                            |   |
101                                            ___
102    For real world applications, you need only to know the connector number (J5 or J6) and
           the pin
103    number to be able to use them.
104 */
105
106 typedef enum{
107     J5,
108     J6
109 }IOExpander;
110
111 typedef enum{
112     OUT,
113     IN
114 }IOPinDirection;
115
116 typedef enum{
117     OFF,
118     ON
119 }IOPinFeature;
120
121 #define IO_PIN_NAME_MAX_LEN 30  //max length for a I/O pin name, including the '\0' string
```

```
122 #define IOEXP_REG_BANK_OFFSET(X) (X>8?1:0) //used to access configuration pins for bank b
        without further logic
123 #define IOEXP_PIN_BIT_OFFSET(X) (X>8?16-X:X-1) //calculates the number of shifts to get the
        pin's bit
124 #define GPIOS_PER_EXPANDER 16 //the number of pins each I/O expander has
125
126
127 //this structures describe a pin in a real aplication in a way an unexperienced user can
        understand
128 typedef struct{
129     IOExpander      exp;
130     BYTE            number;
131     IOPinDirection  dir;
132     IOPinFeature    pullup;
133     IOPinFeature    inverted;
134 }IOPinConfig;
135
136 typedef struct{
137     char            name[IO_PIN_NAME_MAX_LEN];  //unique identifier of GPIO
138     IOPinConfig     config;
139 }IOPinDescription;
140
141 BOOL SetGPIOConfig(const rom char* name, IOPinConfig* config);
142 BOOL GetGPIOConfig(const rom char* name, IOPinConfig* config);
143 BOOL CompareGPIOConfig(IOPinConfig* config1, IOPinConfig* config2);
144 BOOL SetGPIO(const rom char* name);
145 BOOL ResetGPIO(const rom char* name);
146 BOOL ToggleGPIO(const rom char* name);
147 BOOL GetGPIO(const rom char* name, BYTE* value);
148 BOOL GetAllGPIO(BYTE* J5A, BYTE* J5B, BYTE* J6A, BYTE* J6B);
149
150 extern rom const IOPinDescription* GPIOsDescription;
151 extern BYTE ngpios;
152 //-----------------------------------------------------------------------
153 // 16-bit Timer1 runs at 32,768 Hz and is preloaded with 0x8000 in order to
154 // generate periodic 1-sec interrupts.  This timer also supports timeout
155 // delays specified in seconds and ticks (32,768 ticks/sec) which are
156 // specified in a TIME data structure.
157 //-----------------------------------------------------------------------
158 typedef struct
159 {
160     ULONG  secs;
161     WORD ticks;
162 } TIME, *PTIME;
163
164 extern  BYTE    SECS, MINS, HOURS;      // RTC variables
165 extern  ULONG   Seconds;                // Seconds since boot
166 extern  BYTE    ADC0[7];                    // ADC readings AN0..AN6
167 WORD    AdcConvert(WORD Adc);       // ADC reading to millivolts
168 //-----------------------------------------------------------------------
169
170 /* TODO: remove
171 //-----------------------------------------------------------------------
172 // External functions used to customize the Dalf firmware to run other
173 // applications on top of it.
174 //-----------------------------------------------------------------------
175 typedef void (*greeting)(void);
176 typedef BYTE (*cmdExtensionDispatch)(void);
177 typedef void (*serviceIO)(void);
178
179 typedef struct
180 {
181     greeting                GreetingFct;
182     cmdExtensionDispatch    CmdExtensionDispatchFct;
183     serviceIO               ServiceIOFct;
184 }ExternalAppSupportFcts;*/
185
186 //-----------------------------------------------------------------------
187
188
189 ////////////////////////////////////////////////////////////////////////
190 //                  E X T E R N A L     F U N C T I O N S             //
191 //                                                                    //
```

```
192 //      Many (if not all) of these functions reside in <dalf.lib>.          //
193 ////////////////////////////////////////////////////////////////////////////
194 //             ** UNCALLED LIBRARY FUNCTIONS **
195 extern  void    WriteIOExp1(BYTE reg, BYTE data);
196 extern  void    WriteIOExp2(BYTE reg, BYTE data);
197 extern  BYTE    ReadIOExp1(BYTE reg);
198 extern  BYTE    ReadIOExp2(BYTE reg);
199 extern  void    WriteExtEE_Byte(WORD adrs, BYTE dat);
200 extern  BYTE    ReadExtEE_Byte(WORD adrs);
201 extern  void    WriteExtEE_Block(WORD adrs, BYTE *buff, BYTE len);
202 extern  void    ReadExtEE_Block(WORD adrs, BYTE *buff, BYTE len);
203 extern  BYTE    ReadIntEE_Byte(WORD adrs);
204 extern  void    WriteIntEE_Byte(WORD adrs, BYTE dat);
205 extern  void    WriteIntEE_Block(WORD adrs, BYTE *buff, BYTE len);
206 extern  void    ReadIntEE_Block(WORD adrs, BYTE *buff, BYTE len);
207 extern  void    WritePot1(BYTE cmd, BYTE data);
208 extern  void    WritePot2(BYTE cmd, BYTE data);
209 extern  void    GetTime(PTIME pTime);
210
211 //             ** CALLED LIBRARY FUNCTIONS **
212     // Interrupt Handlers
213 extern  void    TMR0_ISR(void);     // TMR0:   Heartbeat
214 extern  void    INT0_ISR(void);     // AB2INT: Mtr2 encoder
215 extern  void    INT1_ISR(void);     // AB1INT: Mtr1 encoder
216 extern  void    TMR1_ISR(void);     // TMR1:   RTC
217 extern  void    TMR2_ISR(void);     // TMR2:   ADC state machine
218 extern  void    INT2_ISR(void);     // I2INT:  Mtr2 current sense
219 extern  void    INT3_ISR(void);     // I1INT:  Mtr1 current sense
220 extern  void    TX1_ISR(void);      // Cmd interface
221 extern  void    RX1_ISR(void);      // Cmd interface
222 extern  void    CCP1_ISR(void);     // EX0 - R/C Channel 1 (Mtr1) Interface
223 extern  void    CCP2_ISR(void);     // EX1 - R/C Channel 2 (Mtr2) Interface
224 extern  void    CCP3_ISR(void);     // EX2 - R/C Channel 3 (Alt) Interface
225 extern  void    SSP2_ISR(void);     // SSP2 - Secondary I2C bus (Dalf Slave).
226
227     // Others
228 extern  void    SystemInit(void);       // Power Up Initialization
229 extern  void    SetDelay(PTIME pTime, ULONG DelaySecs, WORD DelayTicks);
230 extern  int     Timeout(PTIME pTime);   // Check for delay expiration
231 extern  void    UpdateVelocity1(void);  // Update Mtr1 velocity
232 extern  void    PendingCmd1(void);      // If Mtr1 stopped, do cmd
233 extern  void    RampMotor1(void);       // Routine Mtr1 ramping
234 extern  void    Trajectory1(void);      // Generate Mtr1 waypoint
235 extern  void    PID1(void);             // Generate Mtr1 command
236 extern  void    UpdateVelocity2(void);  // Update Mtr2 velocity
237 extern  void    PendingCmd2(void);      // If Mtr2 stopped, do cmd
238 extern  void    RampMotor2(void);       // Routine Mtr2 ramping
239 extern  void    Trajectory2(void);      // Generate Mtr2 waypoint
240 extern  void    PID2(void);             // Generate Mtr2 command
241
242 extern  BYTE    TxChr(char c);          // USART1: XMIT(c)
243 extern  BYTE    TE_CmdParse(void);      // TE.   Parses ASCIZ string in Tx1_Buff
244 extern  BYTE    API_CmdParse(void);     // API.  Parses pkt data in Tx1_Buff
245 extern  BYTE    I2C2_CmdParse(void);    // I2C2. Parses pkt' data in I2C2_Buff
246 extern  BYTE    TeCmdDispatch(void);    // TE.   Cmd Dispatch
247 extern  BYTE    ApiCmdDispatch(void);   // API.  Cmd Dispatch
248 extern  BYTE    I2C2CmdDispatch(void);  // I2C2. Cmd Dispatch
249 extern  void    DoReset(void);          // Microcontroller reset
250 extern  void    DoIdle(void);           // Enter Low Power (IDLE) Mode
251 extern  void    Mtr1FlagFix(void);      // Fixup after OvrCrnt handled by ISR
252 extern  void    Mtr2FlagFix(void);      // Fixup after OvrCrnt handled by ISR
253
254
255 //XXX: this was moved here without though. Analyse this properly
256 int     printf(const rom char *fmt, ...);
257
258 ////////////////////////////////////////////////////////////////////////////
259 //                E X T E N D E D     F U N C T I O N S                    //
260 //                                                                         //
261 //             System functions used because of the ROVIM Project.         //
262 ////////////////////////////////////////////////////////////////////////////
263
264 // system functions
```

```
265 void SystemInitExt(void);
266 #ifdef WATCHDOG_ENABLED
267 void InitWatchdog(void);
268 void KickWatchdog(void);
269 void HardReset(void);
270 #endif
271 DWORD CalculateDelayMs(PTIME start, PTIME end);
272 void EmergencyStopMotors(void);
273 void LockCriticalResourcesAccess(void);
274 void UnlockCriticalResourcesAccess(void);
275 BOOL IsStandardCommandLocked(BYTE cmd);
276 BOOL IsExtendedCommandLocked(BYTE cmd);
277 BYTE TeCmdDispatchExt(void);
278 BYTE I2C2CmdDispatchExt(void);
279 BYTE TeProcessAck(void);
280 BYTE TeDisableAck(void);
281 BYTE TE_CmdParseExt(void);
282
283 void OpenLoopTune1(void);
284 void OpenLoopTune2(void);
285 void SoftStop(BYTE mtr);
286 void MoveMtrOpenLoop(BYTE mtr, BYTE dir, BYTE spd, BYTE slew);
287 void MoveMtrClosedLoop(BYTE mtr, short long tgt, WORD v, WORD a);
288
289 //support functions
290 void DEBUG_PrintCmd(void);
291 void SetVerbosity(BYTE level);
292 BYTE GetVerbosity(void);
293 #ifdef HELP_ENABLED
294 BYTE ShowHelp(void);
295 #endif
296 #ifdef LOG_ENABLED
297 void LOG_LogInit(void);
298 #endif
299
300 void Greeting(void);
301
302 // External switches used for pot control modes
303 #define SWITCH0 (PORTD&0x01)     // PORTD.0: On/Off
304 #define SWITCH1 (PORTD&0x02)     // PORTD.1: Dir (POTF only)
305
306
307 // Oscillator Speed Definition
308 #define FOSC    (D'40000000')    // define FOSC to PICmicro
309
310 //////////////////////////////////////////////
311 //   I2C Device Addresses for On-board Devices  //
312 //////////////////////////////////////////////
313 #define Dev_24LC512      0xA0     // I2C Device Adrs (Microchip 24LC512 EEPROM)
314 #define Dev_MCP23017_1   0x42     // I2C Device Adrs (Microchip MCP23017 I/O EXP)
315 #define Dev_MCP23017_2   0x40     // I2C Device Adrs (Microchip MCP23017 I/O EXP)
316 #define Dev_MAX5478_1    0x50     // I2C Device Adrs (MAXIM, Dual digital pot)
317 #define Dev_MAX5478_2    0x52     // I2C Device Adrs (MAXIM, Dual digital pot)
318 //          Dev_DALF     0x60     ; Dalf as slave on I2C2 bus (See Parm Block)
319
320 //////////////////////////////////////////////
321 //   I2C Register Addresses for MCP23017 Devices  //
322 //                                              //
323 //   Note: Reg#s assume IOCON.BANK=0            //
324 //////////////////////////////////////////////
325 // --- PORT A Registers ---
326 #define IODIRA       0x00     // I/O Direction          ("0"=Output, "1"=Input)
327 #define IPOLA        0x02     // Input Polarity         ("1"=Inverted)
328 #define GPINTENA     0x04     // Int-On-Change Enable   ("0"=Disable, "1"=Enable)
329 #define DEFVALA      0x06     // Default Compare Reg    (for Int-On-Change)
330 #define INTCONA      0x08     // Int Control            ("0"=pin, "1"=DEFVAL)
331 #define IOCON        0x0A     // I/O Config (A & B)
332 //#define  IOCONA      0x0A     // I/O Configuration
333 #define GPPUA        0x0C     // Pullup Enable          ("0"=Disable, "1"=Enable)
334 #define INTFA        0x0E     // Interrupt Flags        ("1"=IntRequest)
335 #define INTCAPA      0x10     // Interrupt Capture      (Pin states on interrupt)
336 #define GPIOA        0x12     // GPIO Port              (Read gives pin states)
337 #define OLATA        0x14     // Output Latch           (Writes drive outputs)
```

```
338 // ---- PORT B Registers ----
339 #define IODIRB        0x01
340 #define IPOLB         0x03
341 #define GPINTENB      0x05
342 #define DEFVALB       0x07
343 #define INTCONB       0x09
344 //#define   IOCONB      0x0B    // Shadow of IOCONA
345 #define GPPUB         0x0D
346 #define INTFB         0x0F
347 #define INTCAPB       0x11
348 #define GPIOB         0x13
349 #define OLATB         0x15
350
351
352 ////////////////////////////////////////////////////////////////////
353 //  I2C Cmds for MAX5478 Devices (Dual Digital 50K Pots) with 256   //
354 //  tap points.  These are Write Only devices with NonVolatile      //
355 //  memory for PwrUp initialization.  There are 4 regs per wiper:   //
356 //                                                                  //
357 //  Reg             Write Effect                                    //
358 //  ----            ------------------------------------------      //
359 //  VREG            Update wiper position only.                     //
360 //  NVREG           Update NVREG only (not wiper or VREG).          //
361 //  NVREGtoVREG     Copy NVREG to VREG & wiper.  No change to NVREG.//
362 //  VREGtoNVREG     Copy VREG to NVREG.  No change to VREG or wiper.//
363 //                                                                  //
364 //  NOTE: Apparently the copy operations don't require a data byte. //
365 ////////////////////////////////////////////////////////////////////
366 // Wiper A
367 #define VREGA             0x11
368 #define NVREGA            0x21
369 #define NVREGtoVREGA      0x61
370 #define VREGtoNVREGA      0x51
371
372 // Wiper B
373 #define VREGB             0x12
374 #define NVREGB            0x22
375 #define NVREGtoVREGB      0x62
376 #define VREGtoNVREGB      0x52
377
378 // Both Wipers A & B
379 #define VREG              0x13
380 #define NVREG             0x23
381 #define NVREGtoVREG       0x63
382 #define VREGtoNVREG       0x53
383
384
385
386
387
388
389 ////////////////////////
390 //    LED Patterns    //
391 ////////////////////////
392 #define LED_FULLOFF 0x00000000          // Always Off
393 #define LED_FAST    0x55555555          // Off for 1, On for 1.
394 #define LED_MED     0x33333333          // Off for 2, On for 2.
395 #define LED_SLOW    0x0F0F0F0F          // Off for 4, On for 4.
396 #define LED_FULLON  0xFFFFFFFF          // Always On
397
398 ////////////////////////
399 //   LED Conditions    //
400 ////////////////////////
401 #define LED_MTR_OFF       LED_FULLOFF   // S=0. ------------------------ Motor off
402 #define LED_MTR_TGA       LED_FAST      // S>0. TGA Active ------------- Moving, PID
403 #define LED_MTR_OPENLP    LED_SLOW      // S>0. TGA Inactive. V>0.----- Moving, OL
404 #define LED_MTR_STALL     LED_FULLON    // S>0. TGA Inactive. V=0.----- Stalled
405 #define LED_OVERCURRENT   LED_FAST      // One or more motors overcurrent
406 #define LED_SIGNAL_LOSS   LED_SLOW      // One or both R/C Signal Loss.
407 #define LED_VBATT         LED_FULLON    // Low VBATT
408 #define ROVIM_T2D_LED_LOCKDOWN     LED_SLOW     // ROVIM is in lockdown
409
410 #define LED_PERIOD  0x64                // 100 msec
```

```
411 #define LED_HYSTERESIS  500             // 500 mV Vbatt Hysteresis for red LED.
412
413 extern  BYTE    LedErr;                     // "1" bits flag err cond'n
414
415 //////////////////////////////////////
416 //  Macros for Fan and LED control  //
417 //////////////////////////////////////
418 //      FAN1    (F.3)
419 #define _FAN1_ON            LATF |=  0x08
420 #define _FAN1_OFF           LATF &= ~0x08
421 #define _FAN1_TOGGLE        LATF ^=  0x08
422
423 //      FAN2    (F.4)
424 #define _FAN2_ON            LATF |=  0x10
425 #define _FAN2_OFF           LATF &= ~0x10
426 #define _FAN2_TOGGLE        LATF ^=  0x10
427
428 //      LED1    (F.6)
429 #define _LED1_ON            LATF |=  0x40
430 #define _LED1_OFF           LATF &= ~0x40
431 #define _LED1_TOGGLE        LATF ^=  0x40
432
433 //      LED2    (F.7)
434 #define _LED2_ON            LATF |=  0x80
435 #define _LED2_OFF           LATF &= ~0x80
436 #define _LED2_TOGGLE        LATF ^=  0x80
437
438 //      LED3    (E.1)
439 #define _LED3_ON            LATE |=  0x02
440 #define _LED3_OFF           LATE &= ~0x02
441 #define _LED3_TOGGLE        LATE ^=  0x02
442
443
444
445 ////////////////////////////////////////////////
446 //    Delay constants for fractional seconds    //
447 //              in SetDelay()                   //
448 //                                              //
449 //        TMR1 tick freq of 32,768 Hz           //
450 ////////////////////////////////////////////////
451 #define usec_31     1       // 31  uSec
452 #define usec_305    10      // 305 uSec
453 #define usec_457    15      // 457 uSec
454
455 #define msec_1      33      //   1 mSec
456 #define msec_5      164     //   5 mSec
457 #define msec_10     328     //  10 mSec
458 #define msec_20     655     //  20 msec
459 #define msec_50     1638    //  50 mSec
460 #define msec_100    3277    // 100 mSec
461 #define msec_200    6554    // 200 msec
462 #define msec_300    9830    // 300 msec
463 #define msec_400    13107   // 400 msec
464 #define msec_500    16384   // 500 mSec
465 #define msec_750    24576   // 750 msec
466 #define sec_1       32768   // 1   sec
467
468 // TIMESVC_REQ Bits
469 #define Vsp1Msk     0x01    // Bit0
470 #define Vsp2Msk     0x02    // Bit1
471 #define PspMsk      0x04    // Bit2
472 #define RcspMsk     0x08    // Bit3
473 #define CmdspMsk    0x10    // Bit4
474
475 //                      DispSvc Bits
476 // DispSvc0 (Low Byte)
477 #define CHRreqMsk       0x01    // Bit0 - xCHR request.
478 #define EreqMsk         0x02    // Bit1 - Motor position request.
479 #define VreqMsk         0x04    // Bit2 - Motor velocity request.
480 #define ADCreqMsk       0x08    // Bit3 - ADC readings request.
481 #define RCreqMsk        0x10    // Bit4 - Radio control request.
482 #define HMSreqMsk       0x20    // Bit5 - Hours:Mins:Secs request.
483 #define MEMBYTEreqMsk   0x40    // Bit6 - Memory Byte request.
```

```
484 #define IOBYTEreqMsk     0x80     // Bit7 – IOEXP Byte request.
485
486 // DispSvc1 (Hi Byte)
487 #define STATreqMsk       0x0100   // Bit8 – Motor Status request.
488 #define PIDreqMsk        0x0200   // Bit9 – PID Motor Parameters request.
489 #define RAMBLKreqMsk     0x0400   // BitA – RAM Block request.
490 #define EXTEEBLKreqMsk   0x0800   // BitB – External EEPROM Block request.
491 #define INTEEBLKreqMsk   0x1000   // BitC – Internal EEPROM Block request.
492 #define RESETreqMsk      0x2000   // BitD – Microcontroller RESET request.
493
494
495 // CmdSource Bits
496 #define      RC_SrcMsk        0x01     // Bit0 – R/C source for CMD/ARG[]
497 #define      POT_SrcMsk       0x02     // Bit1 – Pots source for CMD/ARG[]
498 #define      TE_SrcMsk        0x04     // Bit2 – TE source for CMD/ARG[]
499 #define      API_SrcMsk       0x08     // Bit3 – API source for CMD/ARG[]
500 #define      I2C2_SrcMsk      0x10     // Bit4 – I2C2 source for CMD/ARG[]
501 #define      RS232_SrcMsk     TE_SrcMsk+API_SrcMsk     // Either RS232
502
503 // ISR_Flags Bits – Set and cleared by ISR.  Read only outside ISR!
504 #define      OverC2Msk   0x02          // Bit1 – Mtr2 OverCurrent detected by ISR
505 #define      OverC1Msk   0x01          // Bit0 – Mtr1 OverCurrent detected by ISR
506
507 // I2C2_PktStatus Bits
508 #define      i2c2_AutoMsk     0x80     // Bit7 – I2C2 Pkt[] ready.  Xmit AutoStart
509 #define      i2c2_ERRMsk      0x40     // Bit6 – I2C2 Error.  Clear after reported.
510
511 // Mtr1_Flags1, Mtr2_Flags1 bits
512 #define      cpend_Msk        0x80     // Bit7 – Cmd pending awaiting motor stop
513 #define      ConstantV        0x40     // Bit6 – Constant Velocity (closed loop).
514 #define      OverCurrent_Msk  0x20     // Bit5 – Over current condition
515 #define      sumhoa_Msk       0x10     // Bit4 – PID Err Summation HoldOff active
516 #define      pida_Msk         0x08     // Bit3 – PID motor control active
517 #define      trev_Msk         0x04     // Bit2 – Traj reverse direction required
518 #define      tga_Msk          0x02     // Bit1 – Trajectory generator active
519 #define      triga_Msk        0x01     // Bit0 – Waiting for Closed Loop Trigger
520
521 // Mtrx_Flags2 bits
522 // Mtrx_Flags2 uses more bits than shown here (highest value caught = 0x1E).
523 // Q: Should I use another flag carrier, to be sure i'm not overwritting anything?
524 #define      OL_stepresp      0x80     // Bit7 – "1"=Measuring motor open loop step response
525 #define      _MtrD_mask       0x20     // Bit5 – "1"=Reverse
526 #define      DisableMsk       0x01     // Bit0 – "1"=Disable all Mtr commands
527
528
529 // SYSMODE Bits
530 #define      IdleMsk     0x80     // Bit7 – '1'=Enable Low Power (IDLE) Mode
531 #define      CmdToMsk    0x40     // Bit6 – '1'=Enable Serial Cmd Timeout
532
533 // MTRx_MODE1 (ERAM) Bits
534 #define fanMsk           0x40     // Bit6 – Powerup Fan State (0=Off, 1=On)
535 #define analogfbMsk      0x20     // Bit5 – "1"=Analog feedback (not incr encoder)
536 #define sumhoMsk         0x10     // Bit4 – "1" = PID errsum holdoff !MODE!
537 #define relMsk           0x08     // Bit3 – "1" = Target Relative mode
538 #define trigMsk          0x04     // Bit2 – "1" = Closed Loop Trigger Mode
539 #define aleadsbMsk       0x02     // Bit1 – Mtr2 Encoder:"1" if A Leads B for Fwd
540 #define disactiveloMsk   0x01     // Bit0 – "1" = DIS signal active low
541
542
543 // MTRx_MODE2 (ERAM) Bits
544 #define PotcMsk          0x80     // Bit7 – '1' = POT Centered mode
545 #define PotfMsk          0x40     // Bit6 – '1' = POT + Reversing Switch mode
546 #define RcNrmMsk         0x20     // Bit5 – '1' = R/C Interface mode
547 #define RcMixMsk         0x10     // Bit4 – '1' = R/C Mixing Active
548 #define RcServoMsk       0x08     // Bit3 – '1' = R/C Servo mode
549 #define PotServoMsk      0x04     // Bit2 – '1' = POT Servo mode
550 #define PotMixMsk        0x02     // Bit1 – '1' = POT Mix mode
551                                   // Bit0 – //UNUSED//
552 #define PotMsk       PotcMsk+PotfMsk+PotServoMsk+PotMixMsk
553 #define RcMsk        RcNrmMsk+RcMixMsk+RcServoMsk
554 #define ManualMsk    PotMsk+RcMsk
555
556
```

```
557 // MTRx_MODE3 (ERAM) Bits
558                                     // Bit7 - //UNUSED//
559 #define AnalogDirMsk     0x40      // Bit6 - "1" = Reverse analog encoder direction
560 #define OsmcMsk          0x20      // Bit5 - '1' = Motor PWM complement for rev
561 #define OcieMsk          0x10      // Bit4 - '1' = Over Current Int Enable
562 #define OcFastOffMsk     0x08      // Bit3 - "1" = Immediate OFF on OverCurrent
563 #define VelDecMask       0x04      // Bit2 - "1" = TE disp velocity in decimal
564 #define PosDecMsk        0x02      // Bit1 - "1" = TE disp position in decimal
565 #define VerboseMsk       0x01      // Bit0 - "1" = Verbose PID output enabled.
566
567
568 // SCFG States
569 #define TEcfg        0x00          // Terminal Emulator Interface (RS232)
570 #define APIcfg       0x01          // Application Programming Interface (RS232)
571
572
573 // NewPulse Bits (Inactivity Detect)
574 #define NewRC1msk    0x80      // Bit7
575 #define NewRC2msk    0x40      // Bit6
576 #define NewRC3msk    0x20      // Bit5
577 #define NewRCmsk     (NewRC1msk | NewRC2msk | NewRC3msk)
578
579
580 // LedErr Bits
581 #define VBATTmsk     0x80          // Bit7: Low VBATT voltage
582 #define OC1msk       0x40          // Bit6: Over Current Mtr 1
583 #define OC2msk       0x20          // Bit5: Over Current Mtr 2
584 #define SL1msk       0x10          // Bit4: R/C SignalLoss Mtr1
585 #define SL2msk       0x08          // Bit3: R/C SignalLoss Mtr2
586 #define Lckmsk       0x04          // Bit2: ROVIM is in Lockdown
587
588
589 // Motor Equates
590 #define FORWARD      0x00      // Direction
591 #define REVERSE      0x01      // Direction
592 #define NEUTRAL      0x02      // Movement type
593 #define HILLHOLD     0x03      // Movement type
594 #define SPEEDZERO    0x00      // Speed
595
596 //Step Response
597 #define MAXSAMPLES 0x3e7     //999 - Maximum samples collected when measuring step response
598
599 //Command line extension definitions
600 #define CUSTOM_CMD_ID_OFFSET   16     //identifier of the first custom command (not
        bellonging to the extended dalf application), "G 'X'"
601
602 extern void (*AckCallback)(void);  //pointer to ack callback
603
604 typedef enum{
605     config,
606     set,
607     reset,
608     toggle1,
609     toggle2,
610     get,
611     getOneAtATime1,
612     getOneAtATime2,
613     idle
614 }GPIOTestSM;
615
616 typedef enum{
617     out,
618     outPullup,
619     outInverted,
620     outPullupInverted,
621     in,
622     inPullup,
623     inInverted,
624     inPullupInverted,
625     end
626 }GPIOConfigSM;
627
628 //////////////////////////////////////////////////
```

```
629 //          Other variables declarations          //
630 ////////////////////////////////////////////////////
631
632 #ifdef WATCHDOG_ENABLED
633     extern WORD watchdogcount;
634 #endif
635 extern  BYTE    CMD,ARG[16],ARGN;           // parsed command info
636 extern  BYTE    SCFG;                       // Serial Configuration (1..3)
637 extern  BYTE    Mtr2_Flags2;                // Motor2 flags2
638 extern  BYTE    Mtr2_Flags1;                // Motor2 flags1
639 extern  short long  encode1;                // Mtr1 position encoder
640 extern  short long  V1;                    // Mtr1 Velocity
641 extern  short long  V2;                     // Mtr2 Velocity
642 extern BYTE MTR2_MODE1, MTR2_MODE2, MTR2_MODE3;
643 extern BYTE VMIN1, VMAX1;
644 extern WORD TPR1;
645 extern BYTE VSP1;
646 extern BYTE VSP2;
647 extern WORD ACC2, VMID2;
648 extern  BYTE    S2,Power2;                  // SPD: [0..100%], [0..VMAX2%]
649 extern  BYTE    S1,Power1;                  // SPD: [0..100%], [0..VMAX1%]
650 extern  BYTE    CmdSource;
651
652 // LED variables
653 extern ULONG   ledshift;       // On/Off LED bit shifter (shifted bit is time period)
654 extern ULONG   grn1pattern;    // LED1: On/Off LED bit pattern
655 extern ULONG   grn2pattern;    // LED2: On/Off LED bit pattern
656 extern ULONG   redpattern;     // LED3: On/Off LED bit pattern
657
658 #define TIME_TO_MSEC(x) ((x.secs*1000) + (x.ticks>>5))
659
660 //Verbosity level mask
661 #define VERBOSITY_DISABLED              0x00
662 #define VERBOSITY_LEVEL_ERROR           0x01
663 #define VERBOSITY_LEVEL_WARNING         0x02
664 #define VERBOSITY_LEVEL_STATUS          0x04
665 #define VERBOSITY_LEVEL_DEBUG           0x08
666 #define VERBOSITY_LEVEL_MSG             0x0F //Prints if any of the previous four is enabled
667
668 #define VERBOSITY_USE_CALL_INFO         0x40
669 #define VERBOSITY_USE_TIMESTAMP         0x80
670
671 //Verbosity print macros
672 #define FATAL_ERROR_MSG(ARGS)       do { \
673     BYTE auxVerbosity = GetVerbosity(); \
674     SetVerbosity(VERBOSITY_LEVEL_ERROR | VERBOSITY_USE_CALL_INFO | VERBOSITY_USE_TIMESTAMP);
        \
675     PRINT_VERBOSITY_MSG("FATAL ERROR!!:\t",VERBOSITY_LEVEL_ERROR,ARGS); \
676     SetVerbosity(auxVerbosity); \
677 } while(0)
678 #define ERROR_MSG(ARGS)             PRINT_VERBOSITY_MSG("ERROR!: \t",VERBOSITY_LEVEL_ERROR,
        ARGS)
679 #define WARNING_MSG(ARGS)           PRINT_VERBOSITY_MSG("WARNING:\t",VERBOSITY_LEVEL_WARNING
        ,ARGS)
680 #define STATUS_MSG(ARGS)            PRINT_VERBOSITY_MSG("STATUS: \t",VERBOSITY_LEVEL_STATUS,
        ARGS)
681 #define DEBUG_MSG(ARGS)             PRINT_VERBOSITY_MSG("DEBUG:  \t",VERBOSITY_LEVEL_DEBUG,
        ARGS)
682 #define MSG(ARGS)                   PRINT_VERBOSITY_MSG("",VERBOSITY_LEVEL_MSG,ARGS)
683
684 /* prints and the test module occupy a lot of space. You may reach a point where program
        memory is
685 full. In that case, you may need to not compile some of those traces */
686 #ifdef REMOVE_DEBUG_PRINTS
687 #undefine DEBUG_MSG
688 #endif
689 #ifdef REMOVE_STATUS_PRINTS
690 #undefine STATUS_MSG
691 #endif
692
693 /*Remember that the __LINE__ macro says it is one line above the actual line on the .c,
        because of
694 the 1st line workaround; so we fix it here*/
```

```
695 #define PRINT_VERBOSITY_MSG(TYPE,VERBOSITY_LEVEL, ARGS) do { \
696     if(SCFG != TEcfg) break;\
697     if(GetVerbosity() & VERBOSITY_LEVEL) { \
698         printf(TYPE); \
699         if (GetVerbosity() & VERBOSITY_USE_CALL_INFO) { \
700             printf("File: "__FILE__"; Line:%d:\t",(__LINE__-1)); \
701         } \
702         if (GetVerbosity() & VERBOSITY_USE_TIMESTAMP) { \
703             printf("Elapsed Time: %lu s\t",Seconds); \
704         } \
705     printf(ARGS); \
706     } \
707 } while(0)
708
709 #endif /*__DALF_H*/
```

```
  1 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  2 //                                                                                            x
  3 //      CCCCC    OOOO     NN   N    FFFFF   IIIII    GGGGG       H    H       x
  4 //     C        O    O    N N  N    F         I     G            H    H       x
  5 //     C        O    O    N  N N    FFFF      I     G   GG       HHHHHH       x
  6 //     C        O    O    N   NN    F         I     G    G       H    H       x
  7 //      CCCCC    OOOO     N    N    F       IIIII    GGGG    O   H    H       x
  8 //                                                                                            x
  9 //                                                                                            x
 10 //    <config.h> - "Fuse" settings the Dalf Motor Control Board                               x
 11 //                                                                                            x
 12 //  (c) Copyright 2006 Embedded Electronics LLC, All rights reserved                          x
 13 //xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 14
 15
 16 // ********************************
 17 // *** Set configuration bits  ****
 18 // ********************************
 19 //#pragma config       OSC = INTIO7            // Internal Osc, OSC2=ClockOut, OSC1=RA7
 20 #pragma config  OSC = EC                      // External Oscillator.  OSC2 is Clock Out.
 21 #pragma config  FCMEN = OFF                   // Fail-Safe Clock Monitor disabled.
 22 #pragma config  IESO = OFF                    // Int/Ext Osc Switchover disabled.
 23 #pragma config  PWRT = ON                     // Powerup Timer On.
 24 #pragma config  BOREN = OFF                   // Brown out reset Off
 25 #pragma config  BORV = 0                  // Brown out reset voltage 4.5V
 26 #pragma config  WDT = OFF                    // Watch Dog disabled - it will be activated
        at run time via InitWatchdog()
 27 #pragma config  WDTPS = 512          // Watch Dog timeout= WDTPS * 4 ms
 28 #pragma config  MCLRE = ON                   // MCLR# enabled.
 29 #pragma config  LPT1OSC = OFF          // Low Power Timer1 Operation disabled.
 30 #pragma config  CCP2MX = PORTE         // CCP2 on RE7
 31 #pragma config  STVREN = ON                  // Stack Overflow Reset enabled.
 32 #pragma config  LVP = OFF                    // Low Voltage ICSP off
 33 #pragma config  BBSIZ = BB2K          // Boot Block Size
 34 #pragma config  XINST = OFF                  // Don't install extended instruct set.
 35 #pragma config  DEBUG = ON                   // Remote debug enabled
 36
 37
 38 // Code Protect: [0x18000...0x1FFFF]
 39 #pragma config  CP0 = OFF
 40 #pragma config  CP1 = OFF
 41 #pragma config  CP2 = OFF
 42 #pragma config  CP3 = OFF
 43 #pragma config  CP4 = OFF
 44 #pragma config  CP5 = OFF
 45 #pragma config  CP6 = OFF
 46 #pragma config  CP7 = OFF
 47 #pragma config  CPB = OFF                    // -- boot block
 48 #pragma config  CPD = OFF                    // -- eeprom
 49
 50 // Write Protection Off (all banks)
 51 #pragma config  WRT0 = OFF
 52 #pragma config  WRT1 = OFF
 53 #pragma config  WRT2 = OFF
 54 #pragma config  WRT3 = OFF
 55 #pragma config  WRT4 = OFF
 56 #pragma config  WRT5 = OFF
```

```
57 #pragma config  WRT6 = OFF
58 #pragma config  WRT7 = OFF
59 #pragma config  WRTB = OFF                    // ---- boot block
60 #pragma config  WRTD = OFF                    // ---- eeprom
61
62 // External Blk Tbl Reads Permitted (all banks) ----------------Investigate This with
       Bootloader!!!
63 #pragma config  EBTR0 = OFF
64 #pragma config  EBTR1 = OFF
65 #pragma config  EBTR2 = OFF
66 #pragma config  EBTR3 = OFF
67 #pragma config  EBTR4 = OFF
68 #pragma config  EBTR5 = OFF
69 #pragma config  EBTR6 = OFF
70 #pragma config  EBTR7 = OFF
71 #pragma config  EBTRB = OFF                    // ---- boot block
```

```
 1 /*--------------------------------------------------------------------------
 2  * $Id: p18f6722.h,v 1.4.2.1 2005/07/25 18:23:28 nairnj Exp $
 3  * MPLAB-Cxx  PIC18F6722 processor header
 4  *
 5  * (c) Copyright 1999-2005 Microchip Technology, All rights reserved
 6  *--------------------------------------------------------------------------*/
 7
 8 #ifndef __18F6722_H
 9 #define __18F6722_H
10
11 extern volatile near unsigned char       SSP2CON2;
12 extern volatile near struct {
13   unsigned SEN:1;
14   unsigned RSEN:1;
15   unsigned PEN:1;
16   unsigned RCEN:1;
17   unsigned ACKEN:1;
18   unsigned ACKDT:1;
19   unsigned ACKSTAT:1;
20   unsigned GCEN:1;
21 } SSP2CON2bits;
22 extern volatile near unsigned char       SSP2CON1;
23 extern volatile near struct {
24   unsigned SSPM0:1;
25   unsigned SSPM1:1;
26   unsigned SSPM2:1;
27   unsigned SSPM3:1;
28   unsigned CKP:1;
29   unsigned SSPEN:1;
30   unsigned SSPOV:1;
31   unsigned WCOL:1;
32 } SSP2CON1bits;
33 extern volatile near unsigned char       SSP2STAT;
34 extern volatile near union {
35   struct {
36     unsigned BF:1;
37     unsigned UA:1;
38     unsigned R_W:1;
39     unsigned S:1;
40     unsigned P:1;
41     unsigned D_A:1;
42     unsigned CKE:1;
43     unsigned SMP:1;
44   };
45   struct {
46     unsigned :2;
47     unsigned I2C_READ:1;
48     unsigned I2C_START:1;
49     unsigned I2C_STOP:1;
50     unsigned I2C_DAT:1;
51   };
52   struct {
53     unsigned :2;
54     unsigned NOT_W:1;
55     unsigned :2;
56     unsigned NOT_A:1;
```

```
  };
  struct {
    unsigned :2;
    unsigned NOT_WRITE:1;
    unsigned :2;
    unsigned NOT_ADDRESS:1;
  };
  struct {
    unsigned :2;
    unsigned READ_WRITE:1;
    unsigned :2;
    unsigned DATA_ADDRESS:1;
  };
  struct {
    unsigned :2;
    unsigned R:1;
    unsigned :2;
    unsigned D:1;
  };
} SSP2STATbits;
extern volatile near unsigned char        SSP2ADD;
extern volatile near unsigned char        SSP2BUF;
extern volatile near unsigned char        ECCP2DEL;
extern volatile near union {
  struct {
    unsigned P2DC0:1;
    unsigned P2DC1:1;
    unsigned P2DC2:1;
    unsigned P2DC3:1;
    unsigned P2DC4:1;
    unsigned P2DC5:1;
    unsigned P2DC6:1;
    unsigned P2RSEN:1;
  };
  struct {
    unsigned PDC0:1;
    unsigned PDC1:1;
    unsigned PDC2:1;
    unsigned PDC3:1;
    unsigned PDC4:1;
    unsigned PDC5:1;
    unsigned PDC6:1;
    unsigned PRSEN:1;
  };
} ECCP2DELbits;
extern volatile near unsigned char        ECCP2AS;
extern volatile near union {
  struct {
    unsigned PSS2BD0:1;
    unsigned PSS2BD1:1;
    unsigned PSS2AC0:1;
    unsigned PSS2AC1:1;
    unsigned ECCP2AS0:1;
    unsigned ECCP2AS1:1;
    unsigned ECCP2AS2:1;
    unsigned ECCP2ASE:1;
  };
  struct {
    unsigned PSSBD0:1;
    unsigned PSSBD1:1;
    unsigned PSSAC0:1;
    unsigned PSSAC1:1;
    unsigned ECCPAS0:1;
    unsigned ECCPAS1:1;
    unsigned ECCPAS2:1;
    unsigned ECCPASE:1;
  };
} ECCP2ASbits;
extern volatile near unsigned char        ECCP3DEL;
extern volatile near union {
  struct {
    unsigned P3DC0:1;
    unsigned P3DC1:1;
```

```
130       unsigned P3DC2:1;
131       unsigned P3DC3:1;
132       unsigned P3DC4:1;
133       unsigned P3DC5:1;
134       unsigned P3DC6:1;
135       unsigned P3RSEN:1;
136     };
137     struct {
138       unsigned PDC0:1;
139       unsigned PDC1:1;
140       unsigned PDC2:1;
141       unsigned PDC3:1;
142       unsigned PDC4:1;
143       unsigned PDC5:1;
144       unsigned PDC6:1;
145       unsigned PRSEN:1;
146     };
147 } ECCP3DELbits;
148 extern volatile near unsigned char        ECCP3AS;
149 extern volatile near union {
150     struct {
151       unsigned PSS3BD0:1;
152       unsigned PSS3BD1:1;
153       unsigned PSS3AC0:1;
154       unsigned PSS3AC1:1;
155       unsigned ECCP3AS0:1;
156       unsigned ECCP3AS1:1;
157       unsigned ECCP3AS2:1;
158       unsigned ECCP3ASE:1;
159     };
160     struct {
161       unsigned PSSBD0:1;
162       unsigned PSSBD1:1;
163       unsigned PSSAC0:1;
164       unsigned PSSAC1:1;
165       unsigned ECCPAS0:1;
166       unsigned ECCPAS1:1;
167       unsigned ECCPAS2:1;
168       unsigned ECCPASE:1;
169     };
170 } ECCP3ASbits;
171 extern volatile near unsigned char        RCSTA2;
172 extern volatile near union {
173     struct {
174       unsigned RCD8:1;
175       unsigned :5;
176       unsigned RC9:1;
177     };
178     struct {
179       unsigned :6;
180       unsigned NOT_RC8:1;
181     };
182     struct {
183       unsigned :6;
184       unsigned RC8_9:1;
185     };
186     struct {
187       unsigned RX9D:1;
188       unsigned OERR:1;
189       unsigned FERR:1;
190       unsigned ADDEN:1;
191       unsigned CREN:1;
192       unsigned SREN:1;
193       unsigned RX9:1;
194       unsigned SPEN:1;
195     };
196 } RCSTA2bits;
197 extern volatile near unsigned char        TXSTA2;
198 extern volatile near union {
199     struct {
200       unsigned TX9D:1;
201       unsigned TRMT:1;
202       unsigned BRGH:1;
```

```
203       unsigned SENDB:1;
204       unsigned SYNC:1;
205       unsigned TXEN:1;
206       unsigned TX9:1;
207       unsigned CSRC:1;
208     };
209     struct {
210       unsigned TXD8:1;
211       unsigned :5;
212       unsigned TX8_9:1;
213     };
214     struct {
215       unsigned :6;
216       unsigned NOT_TX8:1;
217     };
218 } TXSTA2bits;
219 extern volatile near unsigned char        TXREG2;
220 extern volatile near unsigned char        RCREG2;
221 extern volatile near unsigned char        SPBRG2;
222 extern volatile near unsigned char        CCP5CON;
223 extern volatile near union {
224     struct {
225       unsigned CCP5M0:1;
226       unsigned CCP5M1:1;
227       unsigned CCP5M2:1;
228       unsigned CCP5M3:1;
229       unsigned DCCP5Y:1;
230       unsigned DCCP5X:1;
231     };
232     struct {
233       unsigned :4;
234       unsigned DC5B0:1;
235       unsigned DC5B1:1;
236     };
237 } CCP5CONbits;
238 extern volatile near unsigned             CCPR5;
239 extern volatile near unsigned char        CCPR5L;
240 extern volatile near unsigned char        CCPR5H;
241 extern volatile near unsigned char        CCP4CON;
242 extern volatile near union {
243     struct {
244       unsigned CCP4M0:1;
245       unsigned CCP4M1:1;
246       unsigned CCP4M2:1;
247       unsigned CCP4M3:1;
248       unsigned DCCP4Y:1;
249       unsigned DCCP4X:1;
250     };
251     struct {
252       unsigned :4;
253       unsigned DC4B0:1;
254       unsigned DC4B1:1;
255     };
256 } CCP4CONbits;
257 extern volatile near unsigned             CCPR4;
258 extern volatile near unsigned char        CCPR4L;
259 extern volatile near unsigned char        CCPR4H;
260 extern volatile near unsigned char        T4CON;
261 extern volatile near struct {
262     unsigned T4CKPS0:1;
263     unsigned T4CKPS1:1;
264     unsigned TMR4ON:1;
265     unsigned T4OUTPS0:1;
266     unsigned T4OUTPS1:1;
267     unsigned T4OUTPS2:1;
268     unsigned T4OUTPS3:1;
269 } T4CONbits;
270 extern volatile near unsigned char        PR4;
271 extern volatile near unsigned char        TMR4;
272 extern volatile near unsigned char        ECCP1DEL;
273 extern volatile near union {
274     struct {
275       unsigned P1DC0:1;
```

```
276       unsigned P1DC1:1;
277       unsigned P1DC2:1;
278       unsigned P1DC3:1;
279       unsigned P1DC4:1;
280       unsigned P1DC5:1;
281       unsigned P1DC6:1;
282       unsigned P1RSEN:1;
283     };
284     struct {
285       unsigned PDC0:1;
286       unsigned PDC1:1;
287       unsigned PDC2:1;
288       unsigned PDC3:1;
289       unsigned PDC4:1;
290       unsigned PDC5:1;
291       unsigned PDC6:1;
292       unsigned PRSEN:1;
293     };
294 } ECCP1DELbits;
295 extern volatile near unsigned char        BAUDCON2;
296 extern volatile near union {
297     struct {
298       unsigned ABDEN:1;
299       unsigned WUE:1;
300       unsigned :1;
301       unsigned BRG16:1;
302       unsigned SCKP:1;
303       unsigned :1;
304       unsigned RCIDL:1;
305       unsigned ABDOVF:1;
306     };
307     struct {
308       unsigned :6;
309       unsigned RCMT:1;
310     };
311 } BAUDCON2bits;
312 extern volatile near unsigned char        SPBRGH2;
313 extern volatile near unsigned char        BAUDCON;
314 extern volatile near union {
315     struct {
316       unsigned ABDEN:1;
317       unsigned WUE:1;
318       unsigned :1;
319       unsigned BRG16:1;
320       unsigned SCKP:1;
321       unsigned :1;
322       unsigned RCIDL:1;
323       unsigned ABDOVF:1;
324     };
325     struct {
326       unsigned :6;
327       unsigned RCMT:1;
328     };
329 } BAUDCONbits;
330 extern volatile near unsigned char        BAUDCON1;
331 extern volatile near union {
332     struct {
333       unsigned ABDEN:1;
334       unsigned WUE:1;
335       unsigned :1;
336       unsigned BRG16:1;
337       unsigned SCKP:1;
338       unsigned :1;
339       unsigned RCIDL:1;
340       unsigned ABDOVF:1;
341     };
342     struct {
343       unsigned :6;
344       unsigned RCMT:1;
345     };
346 } BAUDCON1bits;
347 extern volatile near unsigned char        SPBRGH;
348 extern volatile near unsigned char        SPBRGH1;
```

```
349 extern volatile near unsigned char       PORTA;
350 extern volatile near union {
351   struct {
352     unsigned RA0:1;
353     unsigned RA1:1;
354     unsigned RA2:1;
355     unsigned RA3:1;
356     unsigned RA4:1;
357     unsigned RA5:1;
358     unsigned RA6:1;
359     unsigned RA7:1;
360   };
361   struct {
362     unsigned :2;
363     unsigned VREFM:1;
364     unsigned VREFP:1;
365     unsigned T0CKI:1;
366     unsigned LVDIN:1;
367   };
368   struct {
369     unsigned AN0:1;
370     unsigned AN1:1;
371     unsigned AN2:1;
372     unsigned AN3:1;
373     unsigned :1;
374     unsigned AN4:1;
375   };
376   struct {
377     unsigned :5;
378     unsigned HLVDIN:1;
379   };
380 } PORTAbits;
381 extern volatile near unsigned char       PORTB;
382 extern volatile near union {
383   struct {
384     unsigned RB0:1;
385     unsigned RB1:1;
386     unsigned RB2:1;
387     unsigned RB3:1;
388     unsigned RB4:1;
389     unsigned RB5:1;
390     unsigned RB6:1;
391     unsigned RB7:1;
392   };
393   struct {
394     unsigned INT0:1;
395     unsigned INT1:1;
396     unsigned INT2:1;
397     unsigned INT3:1;
398     unsigned KBI0:1;
399     unsigned KBI1:1;
400     unsigned KBI2:1;
401     unsigned KBI3:1;
402   };
403   struct {
404     unsigned FLT0:1;
405   };
406 } PORTBbits;
407 extern volatile near unsigned char       PORTC;
408 extern volatile near union {
409   struct {
410     unsigned RC0:1;
411     unsigned RC1:1;
412     unsigned RC2:1;
413     unsigned RC3:1;
414     unsigned RC4:1;
415     unsigned RC5:1;
416     unsigned RC6:1;
417     unsigned RC7:1;
418   };
419   struct {
420     unsigned T1OSO:1;
421     unsigned T1OSI:1;
```

```
422       unsigned ECCP1:1;
423       unsigned SCK:1;
424       unsigned SDI:1;
425       unsigned SDO:1;
426       unsigned TX:1;
427       unsigned RX:1;
428     };
429     struct {
430       unsigned T13CKI:1;
431       unsigned ECCP2:1;
432       unsigned :1;
433       unsigned SCL:1;
434       unsigned SDA:1;
435       unsigned :1;
436       unsigned CK:1;
437       unsigned DT:1;
438     };
439     struct {
440       unsigned :1;
441       unsigned CCP2:1;
442       unsigned CCP1:1;
443       unsigned SCL1:1;
444       unsigned SDA1:1;
445       unsigned :1;
446       unsigned CK1:1;
447       unsigned DT1:1;
448     };
449     struct {
450       unsigned :1;
451       unsigned P2A:1;
452       unsigned P1A:1;
453       unsigned SCK1:1;
454       unsigned SDI1:1;
455       unsigned SDO1:1;
456       unsigned TX1:1;
457       unsigned RX1:1;
458     };
459 } PORTCbits;
460 extern volatile near unsigned char        PORTD;
461 extern volatile near union {
462     struct {
463       unsigned RD0:1;
464       unsigned RD1:1;
465       unsigned RD2:1;
466       unsigned RD3:1;
467       unsigned RD4:1;
468       unsigned RD5:1;
469       unsigned RD6:1;
470       unsigned RD7:1;
471     };
472     struct {
473       unsigned PSP0:1;
474       unsigned PSP1:1;
475       unsigned PSP2:1;
476       unsigned PSP3:1;
477       unsigned PSP4:1;
478       unsigned PSP5:1;
479       unsigned PSP6:1;
480       unsigned PSP7:1;
481     };
482     struct {
483       unsigned :5;
484       unsigned SDA2:1;
485       unsigned SCL2:1;
486       unsigned SS2:1;
487     };
488     struct {
489       unsigned :4;
490       unsigned SDO2:1;
491       unsigned SDI2:1;
492       unsigned SCK2:1;
493       unsigned NOT_SS2:1;
494     };
```

```
} PORTDbits;
extern volatile near unsigned char       PORTE;
extern volatile near union {
  struct {
    unsigned RE0:1;
    unsigned RE1:1;
    unsigned RE2:1;
    unsigned RE3:1;
    unsigned RE4:1;
    unsigned RE5:1;
    unsigned RE6:1;
    unsigned RE7:1;
  };
  struct {
    unsigned RD:1;
    unsigned WR:1;
    unsigned CS:1;
    unsigned :4;
    unsigned ECCP2:1;
  };
  struct {
    unsigned NOT_RD:1;
    unsigned NOT_WR:1;
    unsigned NOT_CS:1;
  };
  struct {
    unsigned P2D:1;
    unsigned P2C:1;
    unsigned P2B:1;
    unsigned P3C:1;
    unsigned P3B:1;
    unsigned P1C:1;
    unsigned P1B:1;
    unsigned P2A:1;
  };
  struct {
    unsigned :7;
    unsigned CCP2:1;
  };
} PORTEbits;
extern volatile near unsigned char       PORTF;
extern volatile near union {
  struct {
    unsigned RF0:1;
    unsigned RF1:1;
    unsigned RF2:1;
    unsigned RF3:1;
    unsigned RF4:1;
    unsigned RF5:1;
    unsigned RF6:1;
    unsigned RF7:1;
  };
  struct {
    unsigned AN5:1;
    unsigned AN6:1;
    unsigned AN7:1;
    unsigned AN8:1;
    unsigned AN9:1;
    unsigned AN10:1;
    unsigned AN11:1;
    unsigned SS1:1;
  };
  struct {
    unsigned :1;
    unsigned C2OUT:1;
    unsigned C1OUT:1;
    unsigned :2;
    unsigned CVREF:1;
    unsigned :1;
    unsigned NOT_SS1:1;
  };
} PORTFbits;
extern volatile near unsigned char       PORTG;
```

```
568 extern volatile near union {
569   struct {
570     unsigned RG0:1;
571     unsigned RG1:1;
572     unsigned RG2:1;
573     unsigned RG3:1;
574     unsigned RG4:1;
575     unsigned RG5:1;
576   };
577   struct {
578     unsigned ECCP3:1;
579     unsigned TX2:1;
580     unsigned RX2:1;
581     unsigned CCP4:1;
582     unsigned CCP5:1;
583     unsigned MCLR:1;
584   };
585   struct {
586     unsigned P3A:1;
587     unsigned CK2:1;
588     unsigned DT2:1;
589     unsigned P3D:1;
590     unsigned P1D:1;
591     unsigned NOT_MCLR:1;
592   };
593   struct {
594     unsigned CCP3:1;
595   };
596 } PORTGbits;
597 extern volatile near unsigned char       LATA;
598 extern volatile near struct {
599   unsigned LATA0:1;
600   unsigned LATA1:1;
601   unsigned LATA2:1;
602   unsigned LATA3:1;
603   unsigned LATA4:1;
604   unsigned LATA5:1;
605   unsigned LATA6:1;
606   unsigned LATA7:1;
607 } LATAbits;
608 extern volatile near unsigned char       LATB;
609 extern volatile near struct {
610   unsigned LATB0:1;
611   unsigned LATB1:1;
612   unsigned LATB2:1;
613   unsigned LATB3:1;
614   unsigned LATB4:1;
615   unsigned LATB5:1;
616   unsigned LATB6:1;
617   unsigned LATB7:1;
618 } LATBbits;
619 extern volatile near unsigned char       LATC;
620 extern volatile near struct {
621   unsigned LATC0:1;
622   unsigned LATC1:1;
623   unsigned LATC2:1;
624   unsigned LATC3:1;
625   unsigned LATC4:1;
626   unsigned LATC5:1;
627   unsigned LATC6:1;
628   unsigned LATC7:1;
629 } LATCbits;
630 extern volatile near unsigned char       LATD;
631 extern volatile near struct {
632   unsigned LATD0:1;
633   unsigned LATD1:1;
634   unsigned LATD2:1;
635   unsigned LATD3:1;
636   unsigned LATD4:1;
637   unsigned LATD5:1;
638   unsigned LATD6:1;
639   unsigned LATD7:1;
640 } LATDbits;
```

```
641 extern volatile near unsigned char       LATE;
642 extern volatile near struct {
643   unsigned LATE0:1;
644   unsigned LATE1:1;
645   unsigned LATE2:1;
646   unsigned LATE3:1;
647   unsigned LATE4:1;
648   unsigned LATE5:1;
649   unsigned LATE6:1;
650   unsigned LATE7:1;
651 } LATEbits;
652 extern volatile near unsigned char       LATF;
653 extern volatile near struct {
654   unsigned LATF0:1;
655   unsigned LATF1:1;
656   unsigned LATF2:1;
657   unsigned LATF3:1;
658   unsigned LATF4:1;
659   unsigned LATF5:1;
660   unsigned LATF6:1;
661   unsigned LATF7:1;
662 } LATFbits;
663 extern volatile near unsigned char       LATG;
664 extern volatile near struct {
665   unsigned LATG0:1;
666   unsigned LATG1:1;
667   unsigned LATG2:1;
668   unsigned LATG3:1;
669   unsigned LATG4:1;
670   unsigned LATG5:1;
671 } LATGbits;
672 extern volatile near unsigned char       DDRA;
673 extern volatile near struct {
674   unsigned RA0:1;
675   unsigned RA1:1;
676   unsigned RA2:1;
677   unsigned RA3:1;
678   unsigned RA4:1;
679   unsigned RA5:1;
680   unsigned RA6:1;
681   unsigned RA7:1;
682 } DDRAbits;
683 extern volatile near unsigned char       TRISA;
684 extern volatile near struct {
685   unsigned TRISA0:1;
686   unsigned TRISA1:1;
687   unsigned TRISA2:1;
688   unsigned TRISA3:1;
689   unsigned TRISA4:1;
690   unsigned TRISA5:1;
691   unsigned TRISA6:1;
692   unsigned TRISA7:1;
693 } TRISAbits;
694 extern volatile near unsigned char       DDRB;
695 extern volatile near struct {
696   unsigned RB0:1;
697   unsigned RB1:1;
698   unsigned RB2:1;
699   unsigned RB3:1;
700   unsigned RB4:1;
701   unsigned RB5:1;
702   unsigned RB6:1;
703   unsigned RB7:1;
704 } DDRBbits;
705 extern volatile near unsigned char       TRISB;
706 extern volatile near struct {
707   unsigned TRISB0:1;
708   unsigned TRISB1:1;
709   unsigned TRISB2:1;
710   unsigned TRISB3:1;
711   unsigned TRISB4:1;
712   unsigned TRISB5:1;
713   unsigned TRISB6:1;
```

```
714     unsigned TRISB7:1;
715   } TRISBbits;
716   extern volatile near unsigned char        DDRC;
717   extern volatile near struct {
718     unsigned RC0:1;
719     unsigned RC1:1;
720     unsigned RC2:1;
721     unsigned RC3:1;
722     unsigned RC4:1;
723     unsigned RC5:1;
724     unsigned RC6:1;
725     unsigned RC7:1;
726   } DDRCbits;
727   extern volatile near unsigned char        TRISC;
728   extern volatile near struct {
729     unsigned TRISC0:1;
730     unsigned TRISC1:1;
731     unsigned TRISC2:1;
732     unsigned TRISC3:1;
733     unsigned TRISC4:1;
734     unsigned TRISC5:1;
735     unsigned TRISC6:1;
736     unsigned TRISC7:1;
737   } TRISCbits;
738   extern volatile near unsigned char        DDRD;
739   extern volatile near struct {
740     unsigned RD0:1;
741     unsigned RD1:1;
742     unsigned RD2:1;
743     unsigned RD3:1;
744     unsigned RD4:1;
745     unsigned RD5:1;
746     unsigned RD6:1;
747     unsigned RD7:1;
748   } DDRDbits;
749   extern volatile near unsigned char        TRISD;
750   extern volatile near struct {
751     unsigned TRISD0:1;
752     unsigned TRISD1:1;
753     unsigned TRISD2:1;
754     unsigned TRISD3:1;
755     unsigned TRISD4:1;
756     unsigned TRISD5:1;
757     unsigned TRISD6:1;
758     unsigned TRISD7:1;
759   } TRISDbits;
760   extern volatile near unsigned char        DDRE;
761   extern volatile near struct {
762     unsigned RE0:1;
763     unsigned RE1:1;
764     unsigned RE2:1;
765     unsigned RE3:1;
766     unsigned RE4:1;
767     unsigned RE5:1;
768     unsigned RE6:1;
769     unsigned RE7:1;
770   } DDREbits;
771   extern volatile near unsigned char        TRISE;
772   extern volatile near struct {
773     unsigned TRISE0:1;
774     unsigned TRISE1:1;
775     unsigned TRISE2:1;
776     unsigned TRISE3:1;
777     unsigned TRISE4:1;
778     unsigned TRISE5:1;
779     unsigned TRISE6:1;
780     unsigned TRISE7:1;
781   } TRISEbits;
782   extern volatile near unsigned char        DDRF;
783   extern volatile near struct {
784     unsigned RF0:1;
785     unsigned RF1:1;
786     unsigned RF2:1;
```

```
787   unsigned RF3:1;
788   unsigned RF4:1;
789   unsigned RF5:1;
790   unsigned RF6:1;
791   unsigned RF7:1;
792 } DDRFbits;
793 extern volatile near unsigned char       TRISF;
794 extern volatile near struct {
795   unsigned TRISF0:1;
796   unsigned TRISF1:1;
797   unsigned TRISF2:1;
798   unsigned TRISF3:1;
799   unsigned TRISF4:1;
800   unsigned TRISF5:1;
801   unsigned TRISF6:1;
802   unsigned TRISF7:1;
803 } TRISFbits;
804 extern volatile near unsigned char       DDRG;
805 extern volatile near struct {
806   unsigned RG0:1;
807   unsigned RG1:1;
808   unsigned RG2:1;
809   unsigned RG3:1;
810   unsigned RG4:1;
811 } DDRGbits;
812 extern volatile near unsigned char       TRISG;
813 extern volatile near struct {
814   unsigned TRISG0:1;
815   unsigned TRISG1:1;
816   unsigned TRISG2:1;
817   unsigned TRISG3:1;
818   unsigned TRISG4:1;
819 } TRISGbits;
820 extern volatile near unsigned char       OSCTUNE;
821 extern volatile near struct {
822   unsigned TUN0:1;
823   unsigned TUN1:1;
824   unsigned TUN2:1;
825   unsigned TUN3:1;
826   unsigned TUN4:1;
827   unsigned :1;
828   unsigned PLLEN:1;
829   unsigned INTSRC:1;
830 } OSCTUNEbits;
831 extern volatile near unsigned char       PIE1;
832 extern volatile near union {
833   struct {
834     unsigned TMR1IE:1;
835     unsigned TMR2IE:1;
836     unsigned CCP1IE:1;
837     unsigned SSPIE:1;
838     unsigned TXIE:1;
839     unsigned RCIE:1;
840     unsigned ADIE:1;
841     unsigned PSPIE:1;
842   };
843   struct {
844     unsigned :3;
845     unsigned SSP1IE:1;
846     unsigned TX1IE:1;
847     unsigned RC1IE:1;
848   };
849 } PIE1bits;
850 extern volatile near unsigned char       PIR1;
851 extern volatile near union {
852   struct {
853     unsigned TMR1IF:1;
854     unsigned TMR2IF:1;
855     unsigned CCP1IF:1;
856     unsigned SSPIF:1;
857     unsigned TXIF:1;
858     unsigned RCIF:1;
859     unsigned ADIF:1;
```

```
860       unsigned PSPIF:1;
861     };
862     struct {
863       unsigned :3;
864       unsigned SSP1IF:1;
865       unsigned TX1IF:1;
866       unsigned RC1IF:1;
867     };
868   } PIR1bits;
869   extern volatile near unsigned char        IPR1;
870   extern volatile near union {
871     struct {
872       unsigned TMR1IP:1;
873       unsigned TMR2IP:1;
874       unsigned CCP1IP:1;
875       unsigned SSPIP:1;
876       unsigned TXIP:1;
877       unsigned RCIP:1;
878       unsigned ADIP:1;
879       unsigned PSPIP:1;
880     };
881     struct {
882       unsigned :3;
883       unsigned SSP1IP:1;
884       unsigned TX1IP:1;
885       unsigned RC1IP:1;
886     };
887   } IPR1bits;
888   extern volatile near unsigned char        PIE2;
889   extern volatile near union {
890     struct {
891       unsigned CCP2IE:1;
892       unsigned TMR3IE:1;
893       unsigned LVDIE:1;
894       unsigned BCLIE:1;
895       unsigned EEIE:1;
896       unsigned :1;
897       unsigned CMIE:1;
898       unsigned OSCFIE:1;
899     };
900     struct {
901       unsigned :2;
902       unsigned HLVDIE:1;
903       unsigned BCL1IE:1;
904     };
905   } PIE2bits;
906   extern volatile near unsigned char        PIR2;
907   extern volatile near union {
908     struct {
909       unsigned CCP2IF:1;
910       unsigned TMR3IF:1;
911       unsigned LVDIF:1;
912       unsigned BCLIF:1;
913       unsigned EEIF:1;
914       unsigned :1;
915       unsigned CMIF:1;
916       unsigned OSCFIF:1;
917     };
918     struct {
919       unsigned :2;
920       unsigned HLVDIF:1;
921       unsigned BCL1IF:1;
922     };
923   } PIR2bits;
924   extern volatile near unsigned char        IPR2;
925   extern volatile near union {
926     struct {
927       unsigned CCP2IP:1;
928       unsigned TMR3IP:1;
929       unsigned LVDIP:1;
930       unsigned BCLIP:1;
931       unsigned EEIP:1;
932       unsigned :1;
```

```
933       unsigned CMIP:1;
934       unsigned OSCFIP:1;
935     };
936     struct {
937       unsigned :2;
938       unsigned HLVDIP:1;
939       unsigned BCL1IP:1;
940     };
941   } IPR2bits;
942   extern volatile near unsigned char        PIE3;
943   extern volatile near struct {
944     unsigned CCP3IE:1;
945     unsigned CCP4IE:1;
946     unsigned CCP5IE:1;
947     unsigned TMR4IE:1;
948     unsigned TX2IE:1;
949     unsigned RC2IE:1;
950     unsigned BCL2IE:1;
951     unsigned SSP2IE:1;
952   } PIE3bits;
953   extern volatile near unsigned char        PIR3;
954   extern volatile near struct {
955     unsigned CCP3IF:1;
956     unsigned CCP4IF:1;
957     unsigned CCP5IF:1;
958     unsigned TMR4IF:1;
959     unsigned TX2IF:1;
960     unsigned RC2IF:1;
961     unsigned BCL2IF:1;
962     unsigned SSP2IF:1;
963   } PIR3bits;
964   extern volatile near unsigned char        IPR3;
965   extern volatile near struct {
966     unsigned CCP3IP:1;
967     unsigned CCP4IP:1;
968     unsigned CCP5IP:1;
969     unsigned TMR4IP:1;
970     unsigned TX2IP:1;
971     unsigned RC2IP:1;
972     unsigned BCL2IP:1;
973     unsigned SSP2IP:1;
974   } IPR3bits;
975   extern volatile near unsigned char        EECON1;
976   extern volatile near struct {
977     unsigned RD:1;
978     unsigned WR:1;
979     unsigned WREN:1;
980     unsigned WRERR:1;
981     unsigned FREE:1;
982     unsigned :1;
983     unsigned CFGS:1;
984     unsigned EEPGD:1;
985   } EECON1bits;
986   extern volatile near unsigned char        EECON2;
987   extern volatile near unsigned char        EEDATA;
988   extern volatile near unsigned char        EEADR;
989   extern volatile near unsigned char        EEADRH;
990   extern volatile near unsigned char        RCSTA;
991   extern volatile near union {
992     struct {
993       unsigned RX9D:1;
994       unsigned OERR:1;
995       unsigned FERR:1;
996       unsigned ADDEN:1;
997       unsigned CREN:1;
998       unsigned SREN:1;
999       unsigned RX9:1;
1000      unsigned SPEN:1;
1001    };
1002    struct {
1003      unsigned RCD8:1;
1004      unsigned :5;
1005      unsigned RC9:1;
```

```
1006    };
1007    struct {
1008      unsigned :6;
1009      unsigned NOT_RC8:1;
1010    };
1011    struct {
1012      unsigned :6;
1013      unsigned RC8_9:1;
1014    };
1015  } RCSTAbits;
1016  extern volatile near unsigned char        RCSTA1;
1017  extern volatile near union {
1018    struct {
1019      unsigned RX9D:1;
1020      unsigned OERR:1;
1021      unsigned FERR:1;
1022      unsigned ADDEN:1;
1023      unsigned CREN:1;
1024      unsigned SREN:1;
1025      unsigned RX9:1;
1026      unsigned SPEN:1;
1027    };
1028    struct {
1029      unsigned RCD8:1;
1030      unsigned :5;
1031      unsigned RC9:1;
1032    };
1033    struct {
1034      unsigned :6;
1035      unsigned NOT_RC8:1;
1036    };
1037    struct {
1038      unsigned :6;
1039      unsigned RC8_9:1;
1040    };
1041  } RCSTA1bits;
1042  extern volatile near unsigned char        TXSTA;
1043  extern volatile near union {
1044    struct {
1045      unsigned TX9D:1;
1046      unsigned TRMT:1;
1047      unsigned BRGH:1;
1048      unsigned SENDB:1;
1049      unsigned SYNC:1;
1050      unsigned TXEN:1;
1051      unsigned TX9:1;
1052      unsigned CSRC:1;
1053    };
1054    struct {
1055      unsigned TXD8:1;
1056      unsigned :5;
1057      unsigned TX8_9:1;
1058    };
1059    struct {
1060      unsigned :6;
1061      unsigned NOT_TX8:1;
1062    };
1063  } TXSTAbits;
1064  extern volatile near unsigned char        TXSTA1;
1065  extern volatile near union {
1066    struct {
1067      unsigned TX9D:1;
1068      unsigned TRMT:1;
1069      unsigned BRGH:1;
1070      unsigned SENDB:1;
1071      unsigned SYNC:1;
1072      unsigned TXEN:1;
1073      unsigned TX9:1;
1074      unsigned CSRC:1;
1075    };
1076    struct {
1077      unsigned TXD8:1;
1078      unsigned :5;
```

```
1079      unsigned TX8_9:1;
1080    };
1081    struct {
1082      unsigned :6;
1083      unsigned NOT_TX8:1;
1084    };
1085  } TXSTA1bits;
1086  extern volatile near unsigned char        TXREG;
1087  extern volatile near unsigned char        TXREG1;
1088  extern volatile near unsigned char        RCREG;
1089  extern volatile near unsigned char        RCREG1;
1090  extern volatile near unsigned char        SPBRG;
1091  extern volatile near unsigned char        SPBRG1;
1092  extern volatile near unsigned char        PSPCON;
1093  extern volatile near struct {
1094    unsigned :4;
1095    unsigned PSPMODE:1;
1096    unsigned IBOV:1;
1097    unsigned OBF:1;
1098    unsigned IBF:1;
1099  } PSPCONbits;
1100  extern volatile near unsigned char        T3CON;
1101  extern volatile near union {
1102    struct {
1103      unsigned TMR3ON:1;
1104      unsigned TMR3CS:1;
1105      unsigned T3SYNC:1;
1106      unsigned T3CCP1:1;
1107      unsigned T3CKPS0:1;
1108      unsigned T3CKPS1:1;
1109      unsigned T3CCP2:1;
1110      unsigned RD16:1;
1111    };
1112    struct {
1113      unsigned :2;
1114      unsigned T3INSYNC:1;
1115    };
1116    struct {
1117      unsigned :2;
1118      unsigned NOT_T3SYNC:1;
1119    };
1120  } T3CONbits;
1121  extern volatile near unsigned char        TMR3L;
1122  extern volatile near unsigned char        TMR3H;
1123  extern volatile near unsigned char        CMCON;
1124  extern volatile near struct {
1125    unsigned CM0:1;
1126    unsigned CM1:1;
1127    unsigned CM2:1;
1128    unsigned CIS:1;
1129    unsigned C1INV:1;
1130    unsigned C2INV:1;
1131    unsigned C1OUT:1;
1132    unsigned C2OUT:1;
1133  } CMCONbits;
1134  extern volatile near unsigned char        CVRCON;
1135  extern volatile near struct {
1136    unsigned CVR0:1;
1137    unsigned CVR1:1;
1138    unsigned CVR2:1;
1139    unsigned CVR3:1;
1140    unsigned CVRSS:1;
1141    unsigned CVRR:1;
1142    unsigned CVROE:1;
1143    unsigned CVREN:1;
1144  } CVRCONbits;
1145  extern volatile near unsigned char        ECCP1AS;
1146  extern volatile near union {
1147    struct {
1148      unsigned PSS1BD0:1;
1149      unsigned PSS1BD1:1;
1150      unsigned PSS1AC0:1;
1151      unsigned PSS1AC1:1;
```

```
1152      unsigned ECCP1AS0:1;
1153      unsigned ECCP1AS1:1;
1154      unsigned ECCP1AS2:1;
1155      unsigned ECCP1ASE:1;
1156    };
1157    struct {
1158      unsigned PSSBD0:1;
1159      unsigned PSSBD1:1;
1160      unsigned PSSAC0:1;
1161      unsigned PSSAC1:1;
1162      unsigned ECCPAS0:1;
1163      unsigned ECCPAS1:1;
1164      unsigned ECCPAS2:1;
1165      unsigned ECCPASE:1;
1166    };
1167  } ECCP1ASbits;
1168  extern volatile near unsigned char        CCP3CON;
1169  extern volatile near union {
1170    struct {
1171      unsigned CCP3M0:1;
1172      unsigned CCP3M1:1;
1173      unsigned CCP3M2:1;
1174      unsigned CCP3M3:1;
1175      unsigned DC3B0:1;
1176      unsigned DC3B1:1;
1177      unsigned P3M0:1;
1178      unsigned P3M1:1;
1179    };
1180    struct {
1181      unsigned :4;
1182      unsigned CCP3Y:1;
1183      unsigned CCP3X:1;
1184    };
1185  } CCP3CONbits;
1186  extern volatile near unsigned char        ECCP3CON;
1187  extern volatile near union {
1188    struct {
1189      unsigned CCP3M0:1;
1190      unsigned CCP3M1:1;
1191      unsigned CCP3M2:1;
1192      unsigned CCP3M3:1;
1193      unsigned DC3B0:1;
1194      unsigned DC3B1:1;
1195      unsigned P3M0:1;
1196      unsigned P3M1:1;
1197    };
1198    struct {
1199      unsigned :4;
1200      unsigned CCP3Y:1;
1201      unsigned CCP3X:1;
1202    };
1203  } ECCP3CONbits;
1204  extern volatile near unsigned             CCPR3;
1205  extern volatile near unsigned char        CCPR3L;
1206  extern volatile near unsigned char        CCPR3H;
1207  extern volatile near unsigned char        CCP2CON;
1208  extern volatile near union {
1209    struct {
1210      unsigned CCP2M0:1;
1211      unsigned CCP2M1:1;
1212      unsigned CCP2M2:1;
1213      unsigned CCP2M3:1;
1214      unsigned DC2B0:1;
1215      unsigned DC2B1:1;
1216      unsigned P2M0:1;
1217      unsigned P2M1:1;
1218    };
1219    struct {
1220      unsigned :4;
1221      unsigned CCP2Y:1;
1222      unsigned CCP2X:1;
1223    };
1224  } CCP2CONbits;
```

```
1225 extern volatile near unsigned char       ECCP2CON;
1226 extern volatile near union {
1227   struct {
1228     unsigned CCP2M0:1;
1229     unsigned CCP2M1:1;
1230     unsigned CCP2M2:1;
1231     unsigned CCP2M3:1;
1232     unsigned DC2B0:1;
1233     unsigned DC2B1:1;
1234     unsigned P2M0:1;
1235     unsigned P2M1:1;
1236   };
1237   struct {
1238     unsigned :4;
1239     unsigned CCP2Y:1;
1240     unsigned CCP2X:1;
1241   };
1242 } ECCP2CONbits;
1243 extern volatile near unsigned            CCPR2;
1244 extern volatile near unsigned char       CCPR2L;
1245 extern volatile near unsigned char       CCPR2H;
1246 extern volatile near unsigned char       CCP1CON;
1247 extern volatile near union {
1248   struct {
1249     unsigned CCP1M0:1;
1250     unsigned CCP1M1:1;
1251     unsigned CCP1M2:1;
1252     unsigned CCP1M3:1;
1253     unsigned DC1B0:1;
1254     unsigned DC1B1:1;
1255     unsigned P1M0:1;
1256     unsigned P1M1:1;
1257   };
1258   struct {
1259     unsigned :4;
1260     unsigned CCP1Y:1;
1261     unsigned CCP1X:1;
1262   };
1263 } CCP1CONbits;
1264 extern volatile near unsigned char       ECCP1CON;
1265 extern volatile near union {
1266   struct {
1267     unsigned CCP1M0:1;
1268     unsigned CCP1M1:1;
1269     unsigned CCP1M2:1;
1270     unsigned CCP1M3:1;
1271     unsigned DC1B0:1;
1272     unsigned DC1B1:1;
1273     unsigned P1M0:1;
1274     unsigned P1M1:1;
1275   };
1276   struct {
1277     unsigned :4;
1278     unsigned CCP1Y:1;
1279     unsigned CCP1X:1;
1280   };
1281 } ECCP1CONbits;
1282 extern volatile near unsigned            CCPR1;
1283 extern volatile near unsigned char       CCPR1L;
1284 extern volatile near unsigned char       CCPR1H;
1285 extern volatile near unsigned char       ADCON2;
1286 extern volatile near struct {
1287   unsigned ADCS0:1;
1288   unsigned ADCS1:1;
1289   unsigned ADCS2:1;
1290   unsigned ACQT0:1;
1291   unsigned ACQT1:1;
1292   unsigned ACQT2:1;
1293   unsigned :1;
1294   unsigned ADFM:1;
1295 } ADCON2bits;
1296 extern volatile near unsigned char       ADCON1;
1297 extern volatile near struct {
```

```
1298    unsigned PCFG0:1;
1299    unsigned PCFG1:1;
1300    unsigned PCFG2:1;
1301    unsigned PCFG3:1;
1302    unsigned VCFG0:1;
1303    unsigned VCFG1:1;
1304  } ADCON1bits;
1305  extern volatile near unsigned char        ADCON0;
1306  extern volatile near union {
1307    struct {
1308      unsigned :1;
1309      unsigned DONE:1;
1310    };
1311    struct {
1312      unsigned :1;
1313      unsigned GO_DONE:1;
1314    };
1315    struct {
1316      unsigned ADON:1;
1317      unsigned GO:1;
1318      unsigned CHS0:1;
1319      unsigned CHS1:1;
1320      unsigned CHS2:1;
1321      unsigned CHS3:1;
1322    };
1323    struct {
1324      unsigned :1;
1325      unsigned NOT_DONE:1;
1326    };
1327  } ADCON0bits;
1328  extern volatile near unsigned             ADRES;
1329  extern volatile near unsigned char        ADRESL;
1330  extern volatile near unsigned char        ADRESH;
1331  extern volatile near unsigned char        SSP1CON2;
1332  extern volatile near struct {
1333    unsigned SEN:1;
1334    unsigned RSEN:1;
1335    unsigned PEN:1;
1336    unsigned RCEN:1;
1337    unsigned ACKEN:1;
1338    unsigned ACKDT:1;
1339    unsigned ACKSTAT:1;
1340    unsigned GCEN:1;
1341  } SSP1CON2bits;
1342  extern volatile near unsigned char        SSPCON2;
1343  extern volatile near struct {
1344    unsigned SEN:1;
1345    unsigned RSEN:1;
1346    unsigned PEN:1;
1347    unsigned RCEN:1;
1348    unsigned ACKEN:1;
1349    unsigned ACKDT:1;
1350    unsigned ACKSTAT:1;
1351    unsigned GCEN:1;
1352  } SSPCON2bits;
1353  extern volatile near unsigned char        SSP1CON1;
1354  extern volatile near struct {
1355    unsigned SSPM0:1;
1356    unsigned SSPM1:1;
1357    unsigned SSPM2:1;
1358    unsigned SSPM3:1;
1359    unsigned CKP:1;
1360    unsigned SSPEN:1;
1361    unsigned SSPOV:1;
1362    unsigned WCOL:1;
1363  } SSP1CON1bits;
1364  extern volatile near unsigned char        SSPCON1;
1365  extern volatile near struct {
1366    unsigned SSPM0:1;
1367    unsigned SSPM1:1;
1368    unsigned SSPM2:1;
1369    unsigned SSPM3:1;
1370    unsigned CKP:1;
```

```
1371    unsigned SSPEN:1;
1372    unsigned SSPOV:1;
1373    unsigned WCOL:1;
1374  } SSPCON1bits;
1375  extern volatile near unsigned char        SSP1STAT;
1376  extern volatile near union {
1377    struct {
1378      unsigned BF:1;
1379      unsigned UA:1;
1380      unsigned R_W:1;
1381      unsigned S:1;
1382      unsigned P:1;
1383      unsigned D_A:1;
1384      unsigned CKE:1;
1385      unsigned SMP:1;
1386    };
1387    struct {
1388      unsigned :2;
1389      unsigned I2C_READ:1;
1390      unsigned I2C_START:1;
1391      unsigned I2C_STOP:1;
1392      unsigned I2C_DAT:1;
1393    };
1394    struct {
1395      unsigned :2;
1396      unsigned NOT_W:1;
1397      unsigned :2;
1398      unsigned NOT_A:1;
1399    };
1400    struct {
1401      unsigned :2;
1402      unsigned NOT_WRITE:1;
1403      unsigned :2;
1404      unsigned NOT_ADDRESS:1;
1405    };
1406    struct {
1407      unsigned :2;
1408      unsigned READ_WRITE:1;
1409      unsigned :2;
1410      unsigned DATA_ADDRESS:1;
1411    };
1412    struct {
1413      unsigned :2;
1414      unsigned R:1;
1415      unsigned :2;
1416      unsigned D:1;
1417    };
1418  } SSP1STATbits;
1419  extern volatile near unsigned char        SSPSTAT;
1420  extern volatile near union {
1421    struct {
1422      unsigned BF:1;
1423      unsigned UA:1;
1424      unsigned R_W:1;
1425      unsigned S:1;
1426      unsigned P:1;
1427      unsigned D_A:1;
1428      unsigned CKE:1;
1429      unsigned SMP:1;
1430    };
1431    struct {
1432      unsigned :2;
1433      unsigned I2C_READ:1;
1434      unsigned I2C_START:1;
1435      unsigned I2C_STOP:1;
1436      unsigned I2C_DAT:1;
1437    };
1438    struct {
1439      unsigned :2;
1440      unsigned NOT_W:1;
1441      unsigned :2;
1442      unsigned NOT_A:1;
1443    };
```

```
1444    struct {
1445      unsigned :2;
1446      unsigned NOT_WRITE:1;
1447      unsigned :2;
1448      unsigned NOT_ADDRESS:1;
1449    };
1450    struct {
1451      unsigned :2;
1452      unsigned READ_WRITE:1;
1453      unsigned :2;
1454      unsigned DATA_ADDRESS:1;
1455    };
1456    struct {
1457      unsigned :2;
1458      unsigned R:1;
1459      unsigned :2;
1460      unsigned D:1;
1461    };
1462  } SSPSTATbits;
1463  extern volatile near unsigned char        SSP1ADD;
1464  extern volatile near unsigned char        SSPADD;
1465  extern volatile near unsigned char        SSP1BUF;
1466  extern volatile near unsigned char        SSPBUF;
1467  extern volatile near unsigned char        T2CON;
1468  extern volatile near struct {
1469    unsigned T2CKPS0:1;
1470    unsigned T2CKPS1:1;
1471    unsigned TMR2ON:1;
1472    unsigned T2OUTPS0:1;
1473    unsigned T2OUTPS1:1;
1474    unsigned T2OUTPS2:1;
1475    unsigned T2OUTPS3:1;
1476  } T2CONbits;
1477  extern volatile near unsigned char        PR2;
1478  extern volatile near unsigned char        TMR2;
1479  extern volatile near unsigned char        T1CON;
1480  extern volatile near union {
1481    struct {
1482      unsigned TMR1ON:1;
1483      unsigned TMR1CS:1;
1484      unsigned T1SYNC:1;
1485      unsigned T1OSCEN:1;
1486      unsigned T1CKPS0:1;
1487      unsigned T1CKPS1:1;
1488      unsigned T1RUN:1;
1489      unsigned RD16:1;
1490    };
1491    struct {
1492      unsigned :2;
1493      unsigned T1INSYNC:1;
1494    };
1495    struct {
1496      unsigned :2;
1497      unsigned NOT_T1SYNC:1;
1498    };
1499  } T1CONbits;
1500  extern volatile near unsigned char        TMR1L;
1501  extern volatile near unsigned char        TMR1H;
1502  extern volatile near unsigned char        RCON;
1503  extern volatile near union {
1504    struct {
1505      unsigned NOT_BOR:1;
1506      unsigned NOT_POR:1;
1507      unsigned NOT_PD:1;
1508      unsigned NOT_TO:1;
1509      unsigned NOT_RI:1;
1510      unsigned :1;
1511      unsigned SBOREN:1;
1512      unsigned IPEN:1;
1513    };
1514    struct {
1515      unsigned BOR:1;
1516      unsigned POR:1;
```

```
1517      unsigned PD:1;
1518      unsigned TO:1;
1519      unsigned RI:1;
1520    };
1521  } RCONbits;
1522  extern volatile near unsigned char        WDTCON;
1523  extern volatile near union {
1524    struct {
1525      unsigned SWDTE:1;
1526    };
1527    struct {
1528      unsigned SWDTEN:1;
1529    };
1530  } WDTCONbits;
1531  extern volatile near unsigned char        HLVDCON;
1532  extern volatile near union {
1533    struct {
1534      unsigned LVDL0:1;
1535      unsigned LVDL1:1;
1536      unsigned LVDL2:1;
1537      unsigned LVDL3:1;
1538      unsigned LVDEN:1;
1539      unsigned IRVST:1;
1540    };
1541    struct {
1542      unsigned LVV0:1;
1543      unsigned LVV1:1;
1544      unsigned LVV2:1;
1545      unsigned LVV3:1;
1546      unsigned :1;
1547      unsigned BGST:1;
1548    };
1549    struct {
1550      unsigned HLVDL0:1;
1551      unsigned HLVDL1:1;
1552      unsigned HLVDL2:1;
1553      unsigned HLVDL3:1;
1554      unsigned HLVDEN:1;
1555      unsigned :2;
1556      unsigned VDIRMAG:1;
1557    };
1558    struct {
1559      unsigned :5;
1560      unsigned IVRST:1;
1561    };
1562  } HLVDCONbits;
1563  extern volatile near unsigned char        LVDCON;
1564  extern volatile near union {
1565    struct {
1566      unsigned LVDL0:1;
1567      unsigned LVDL1:1;
1568      unsigned LVDL2:1;
1569      unsigned LVDL3:1;
1570      unsigned LVDEN:1;
1571      unsigned IRVST:1;
1572    };
1573    struct {
1574      unsigned LVV0:1;
1575      unsigned LVV1:1;
1576      unsigned LVV2:1;
1577      unsigned LVV3:1;
1578      unsigned :1;
1579      unsigned BGST:1;
1580    };
1581    struct {
1582      unsigned HLVDL0:1;
1583      unsigned HLVDL1:1;
1584      unsigned HLVDL2:1;
1585      unsigned HLVDL3:1;
1586      unsigned HLVDEN:1;
1587      unsigned :2;
1588      unsigned VDIRMAG:1;
1589    };
```

```
1590    struct {
1591      unsigned :5;
1592      unsigned IVRST:1;
1593    };
1594  } LVDCONbits;
1595  extern volatile near unsigned char        OSCCON;
1596  extern volatile near union {
1597    struct {
1598      unsigned SCS0:1;
1599      unsigned SCS1:1;
1600      unsigned IOFS:1;
1601      unsigned OSTS:1;
1602      unsigned IRCF0:1;
1603      unsigned IRCF1:1;
1604      unsigned IRCF2:1;
1605      unsigned IDLEN:1;
1606    };
1607    struct {
1608      unsigned :2;
1609      unsigned FLTS:1;
1610    };
1611  } OSCCONbits;
1612  extern volatile near unsigned char        T0CON;
1613  extern volatile near union {
1614    struct {
1615      unsigned T0PS0:1;
1616      unsigned T0PS1:1;
1617      unsigned T0PS2:1;
1618      unsigned PSA:1;
1619      unsigned T0SE:1;
1620      unsigned T0CS:1;
1621      unsigned T08BIT:1;
1622      unsigned TMR0ON:1;
1623    };
1624    struct {
1625      unsigned :3;
1626      unsigned T0PS3:1;
1627    };
1628  } T0CONbits;
1629  extern volatile near unsigned char        TMR0L;
1630  extern volatile near unsigned char        TMR0H;
1631  extern           near unsigned char        STATUS;
1632  extern           near struct {
1633    unsigned C:1;
1634    unsigned DC:1;
1635    unsigned Z:1;
1636    unsigned OV:1;
1637    unsigned N:1;
1638  } STATUSbits;
1639  extern           near unsigned             FSR2;
1640  extern           near unsigned char        FSR2L;
1641  extern           near unsigned char        FSR2H;
1642  extern volatile near unsigned char        PLUSW2;
1643  extern volatile near unsigned char        PREINC2;
1644  extern volatile near unsigned char        POSTDEC2;
1645  extern volatile near unsigned char        POSTINC2;
1646  extern           near unsigned char        INDF2;
1647  extern           near unsigned char        BSR;
1648  extern           near unsigned             FSR1;
1649  extern           near unsigned char        FSR1L;
1650  extern           near unsigned char        FSR1H;
1651  extern volatile near unsigned char        PLUSW1;
1652  extern volatile near unsigned char        PREINC1;
1653  extern volatile near unsigned char        POSTDEC1;
1654  extern volatile near unsigned char        POSTINC1;
1655  extern           near unsigned char        INDF1;
1656  extern           near unsigned char        WREG;
1657  extern           near unsigned             FSR0;
1658  extern           near unsigned char        FSR0L;
1659  extern           near unsigned char        FSR0H;
1660  extern volatile near unsigned char        PLUSW0;
1661  extern volatile near unsigned char        PREINC0;
1662  extern volatile near unsigned char        POSTDEC0;
```

```
1663 extern volatile near unsigned char        POSTINC0;
1664 extern          near unsigned char        INDF0;
1665 extern volatile near unsigned char        INTCON3;
1666 extern volatile near union {
1667   struct {
1668     unsigned INT1F:1;
1669     unsigned INT2F:1;
1670     unsigned INT3F:1;
1671     unsigned INT1E:1;
1672     unsigned INT2E:1;
1673     unsigned INT3E:1;
1674     unsigned INT1P:1;
1675     unsigned INT2P:1;
1676   };
1677   struct {
1678     unsigned INT1IF:1;
1679     unsigned INT2IF:1;
1680     unsigned INT3IF:1;
1681     unsigned INT1IE:1;
1682     unsigned INT2IE:1;
1683     unsigned INT3IE:1;
1684     unsigned INT1IP:1;
1685     unsigned INT2IP:1;
1686   };
1687 } INTCON3bits;
1688 extern volatile near unsigned char        INTCON2;
1689 extern volatile near union {
1690   struct {
1691     unsigned RBIP:1;
1692     unsigned INT3P:1;
1693     unsigned T0IP:1;
1694     unsigned INTEDG3:1;
1695     unsigned INTEDG2:1;
1696     unsigned INTEDG1:1;
1697     unsigned INTEDG0:1;
1698     unsigned NOT_RBPU:1;
1699   };
1700   struct {
1701     unsigned :1;
1702     unsigned INT3IP:1;
1703     unsigned TMR0IP:1;
1704     unsigned :4;
1705     unsigned RBPU:1;
1706   };
1707 } INTCON2bits;
1708 extern volatile near unsigned char        INTCON;
1709 extern volatile near union {
1710   struct {
1711     unsigned RBIF:1;
1712     unsigned INT0F:1;
1713     unsigned T0IF:1;
1714     unsigned RBIE:1;
1715     unsigned INT0E:1;
1716     unsigned T0IE:1;
1717     unsigned PEIE:1;
1718     unsigned GIE:1;
1719   };
1720   struct {
1721     unsigned :1;
1722     unsigned INT0IF:1;
1723     unsigned TMR0IF:1;
1724     unsigned :1;
1725     unsigned INT0IE:1;
1726     unsigned TMR0IE:1;
1727     unsigned GIEL:1;
1728     unsigned GIEH:1;
1729   };
1730 } INTCONbits;
1731 extern          near unsigned             PROD;
1732 extern          near unsigned char        PRODL;
1733 extern          near unsigned char        PRODH;
1734 extern volatile near unsigned char        TABLAT;
1735 extern volatile near unsigned short long TBLPTR;
```

```
1736 extern volatile near unsigned char       TBLPTRL;
1737 extern volatile near unsigned char       TBLPTRH;
1738 extern volatile near unsigned char       TBLPTRU;
1739 extern volatile near unsigned short long PC;
1740 extern volatile near unsigned char       PCL;
1741 extern volatile near unsigned char       PCLATH;
1742 extern volatile near unsigned char       PCLATU;
1743 extern volatile near unsigned char       STKPTR;
1744 extern volatile near union {
1745   struct {
1746     unsigned STKPTR0:1;
1747     unsigned STKPTR1:1;
1748     unsigned STKPTR2:1;
1749     unsigned STKPTR3:1;
1750     unsigned STKPTR4:1;
1751     unsigned :1;
1752     unsigned STKUNF:1;
1753     unsigned STKFUL:1;
1754   };
1755   struct {
1756     unsigned SP0:1;
1757     unsigned SP1:1;
1758     unsigned SP2:1;
1759     unsigned SP3:1;
1760     unsigned SP4:1;
1761     unsigned :2;
1762     unsigned STKOVF:1;
1763   };
1764 } STKPTRbits;
1765 extern           near unsigned short long TOS;
1766 extern           near unsigned char       TOSL;
1767 extern           near unsigned char       TOSH;
1768 extern           near unsigned char       TOSU;
1769
1770
1771 /*-----------------------------------------------------------------------
1772  * Some useful defines for inline assembly stuff
1773  *-----------------------------------------------------------------------*/
1774 #define ACCESS 0
1775 #define BANKED 1
1776
1777 /*-----------------------------------------------------------------------
1778  * Some useful macros for inline assembly stuff
1779  *-----------------------------------------------------------------------*/
1780 #define Nop()     {_asm nop _endasm}
1781 #define ClrWdt()  {_asm clrwdt _endasm}
1782 #define Sleep()   {_asm sleep _endasm}
1783 #define Reset()   {_asm reset _endasm}
1784
1785 #define Rlcf(f,dest,access)  {_asm movlb f rlcf f,dest,access _endasm}
1786 #define Rlncf(f,dest,access) {_asm movlb f rlncf f,dest,access _endasm}
1787 #define Rrcf(f,dest,access)  {_asm movlb f rrcf f,dest,access _endasm}
1788 #define Rrncf(f,dest,access) {_asm movlb f rrncf f,dest,access _endasm}
1789 #define Swapf(f,dest,access) {_asm movlb f swapf f,dest,access _endasm }
1790
1791 /*-----------------------------------------------------------------------
1792  * A fairly inclusive set of registers to save for interrupts.
1793  * These are locations which are commonly used by the compiler.
1794  *-----------------------------------------------------------------------*/
1795 #define INTSAVELOCS TBLPTR, TABLAT, PROD
1796
1797
1798 #endif
```

```
1 #line 1 "rovim_t2d.c"          //work around the __FILE__ screwup on windows, http://www.
      microchip.com/forums/m746272.aspx
2 //cannot set breakpoints if this directive is used:
3 //info: http://www.microchip.com/forums/m105540-print.aspx
4 //uncomment only when breakpoints are no longer needed
5 /*****************************************************************************
6 *****************************************************************************
7 **
8 **
```

```
 9 **                  rovim_t2d.c - "tracção, travagem e direcção" (T2D)
10 **                        module of the ROVIM project.
11 **
12 **       This module builds on and extends the firmware of the Dalf-1F motor
13 **       control board to implement the T2D module of the ROVIM project.
14 **
15 **       This code was designed originally for the Dalf-1F motor control
16 **       board, the brain of the T2D module.
17 **       Original Dalf-1F firmware revision was 1.73.
18 **       See Dalf-1F owner's manual and the ROVIM T2D documentation for more
19 **       details.
20 **
21 **           The ROVIM Project
22 ********************************************************************************
23 ********************************************************************************/
24
25 #include "p18f6722.h"
26 #include "rovim.h"
27 #include "dalf.h"
28 #include <stdio.h>
29 #include <string.h>
30
31 #pragma romdata DefaultGPIOsDescription
32 /* Since MCC18 cannot create RAM objects larger than 256 bytes, we must create a specific
       section
33 for this data structure. This was done in ROM, since this is only read-only data.
34 Also, storing the GPIO name in ROM adds a lot of convenience when calling driver functions*/
35 rom const IOPinDescription DefaultGPIOsDescription[]={
36     /*name,                          exp, number,   dir,   pullup, inverted*/
37     /*controls progressive braking on SigmaDrive, when on auto mode. Used as PWM switch to
           feed
38 analogue input. This should be used to slow down the vehicle during regular operation.*/
39     { "decelerator",                 { J5,    1,      OUT,    OFF,   OFF }},
40     /*detects when electric brake has reached unclamp side end-of-travel switch. Logic '1'
           indicates
41  the electric brake is fully unclamped.*/
42     { "brake unclamp switch",        { J5,    2,      IN,     OFF,   ON  }},
43     /*controls accelerator on SigmaDrive, when on auto mode. Used as PWM switch to feed
           analogue input.*/
44     { "accelerator",                 { J5,    3,      OUT,    OFF,   OFF }},
45     /*detects when the SigmaDrive B+ pin is being powered. Logic '1' indicates the
           SigmaDrive
46 is turned on and the fuse and contactor are OK.*/
47     { "traction voltage sensor",     { J5,    4,      IN,     OFF,   OFF }},
48     /*controls SigmaDrive foot switch, when on auto mode. Active low: logic '0' means the
           switch is
49 pressed and the vehicle is ready to move.*/
50     { "activate traction",           { J5,    5,      OUT,    OFF,   OFF }},
51     /*detects when electric brake has reached clamp side end-of-travel switch. Logic '1'
           indicates
52 the electric brake is fully clamped and a lockdown cause (emergency switch, dead man trigger
       ,
53 etc.) is active.*/
54     { "brake clamp switch",          { J5,    6,      IN,     OFF,   OFF }},
55     /*controls electric brake unclamp. To properly unclamp electric brake, this signal has
           to be
56 active until unclamp side end-of-travel is reached.*/
57     { "brake unclamper",             { J5,    7,      OUT,    OFF,   OFF }},
58     /*detects when direction position reaches either of the side end-of-travel switches.
           Logic '1'
59 means end-of-travel is switched on and direction is on an erroneous state.*/
60     { "direction error switch",      { J5,    8,      IN,     OFF,   OFF }},
61     /*controls reverse switch on SigmaDrive, when on auto mode. Active low: logic '0' means
           the
62  switch is pressed and reverse drive is engaged. This switch cannot be active simultaneously
        with
63  "engage forward", pin 11.*/
64     { "engage reverse",              { J5,    9,      OUT,    OFF,   OFF }},
65     /*detects when there is a command to unclamp the brake, either by sw or hw. This may
           regardless
66 off the brake clamp/unclamp switches state. Logic '1' indicates there is such a command.*/
67     { "brake unclamp command",       { J5,    10,     IN,     OFF,   OFF }},
```

```
68      /*controls forward switch on SigmaDrive, when on auto mode. Active low: logic '0' means
            the
69   switch is pressed and forward drive is engaged. This switch cannot be active simultaneously
         with
70   "engage reverse", pin 9.*/
71      { "engage forward",                 { J5,    11,     OUT,    OFF,   OFF }},
72      /*detects when manual/auto switch is on auto mode. When auto mode is engaged, the
73   control signals to the SigmaDrive controller must be provided by this program. Logic '1'
         means auto
74   mode is engaged*/
75      { "auto mode switch",               { J5,    12,     IN,     OFF,   OFF }},
76      /*controls electric brake clamping. To brake electric brake, only a pulse from this
            signal is
77   needed.*/
78      { "brake clamper",                  { J5,    13,     OUT,    OFF,   OFF }},
79      /*detects an emergency stop condition, regarless of the source. Logic '1' indicates the
            switch
80   is pressed and the brake is trying to clamp.*/
81      { "emergency stop condition",       { J5,    14,     IN,     OFF,   OFF }},
82      /*controls traction controller handbrake feature. Logic '1' means the handbrake is
            engaged and
83   the vehicle will hold its position.*/
84      { "engage handbrake",               { J5,    15,     OUT,    OFF,   OFF }},
85      /*Unused*/
86      { "",                               { J5,    16,     IN,     OFF,   OFF }},
87   };
88
89   const rom BYTE nDefaultgpios=(BYTE) (sizeof(DefaultGPIOsDescription)/sizeof(
         DefaultGPIOsDescription[0]));
90   #pragma romdata //resume rom data allocation on the standard section
91
92   static BOOL ResourcesLockFlag=FALSE;
93   static const BYTE ROVIM_T2D_CommandsToAllow[]={
94       ROVIM_T2D_LOCKDOWN_CMD_CODE,
95       ROVIM_T2D_RELEASE_CMD_CODE,
96       ROVIM_T2D_SOFTSTOP_CMD_CODE,
97       ROVIM_T2D_ACCELERATE_CMD_CODE,
98       ROVIM_T2D_DECELERATE_CMD_CODE,
99       ROVIM_T2D_SET_MOVEMENT_CMD_CODE,
100      ROVIM_T2D_DEBUG_CTRL_CMD_CODE
101      };
102  static const BYTE ROVIM_T2D_nCommandsToAllow=(BYTE) (sizeof(ROVIM_T2D_CommandsToAllow)/
         sizeof(ROVIM_T2D_CommandsToAllow[0]));
103
104  WORD     ROVIM_T2D_sysmonitorcount;              // ROVIM T2D system state monitoring timeout
         counter;
105  WORD     ROVIM_T2D_pwmrefreshcount;              // ROVIM T2D PWM refresh timeout counter;
106  BOOL     ManualSysMonitoring=FALSE;
107  BYTE     DebugPWM=0;
108
109  //The pointer to easily switch configurations
110  rom const IOPinDescription* GPIOsDescription = DefaultGPIOsDescription;
111  BYTE ngpios=0;
112
113  //Duty cycle for the traction accelerador and decelerator
114  static BYTE AccDutyCycle=0;
115  static BYTE DccDutyCycle=0;
116  static unsigned int WaitForAccDccDecay=0;
117  static BYTE PeriodCnt=0;
118
119  static BOOL ResetPrintFlags=FALSE;
120
121  //Brake locl/unlock flags
122  static BOOL UnlockingBrake=FALSE;
123  BOOL inLockdown=FALSE;
124  BOOL autoMode=FALSE;
125  BOOL SigmaDError=FALSE;
126  static WORD MotorStressMonitor[3]={0,0,0};
127
128  //vehicle movement description
129  static long settlingTime=0;
130  static movement desiredMovement={0};
131  BYTE movementType=HILLHOLD;
```

```
132 long /*acc1=0,*/vel1=0;
133 BOOL ForcePrintMsg=FALSE;
134
135 #if 0
136 static const BYTE SpeedToDutyCycleNoLoadLUTSigmaDConf1[50]=
137 /*Speed (Km/10/h):*/
138 /*duty cycle (%):*/{0,};
139 static const BYTE *SpeedToDutyCycle;
140 static const BYTE SpeedToDutyCycleLen=0;
141 #endif
142 static BYTE SpeedDCScaling=12;  //equals to 1
143
144 //configure basic ROVIM features needed early on. To be called as soon as possible
145 void ROVIM_T2D_Init(void)
146 {
147     ROVIM_T2D_sysmonitorcount=-1; //"disable" IO exp sampling for now
148     SetVerbosity (INIT_VERBOSITY_LEVEL);
149     ROVIM_T2D_ConfigSerialPort();
150     ROVIM_T2D_ConfigGPIOs();
151     ROVIM_T2D_LockUnusedResourcesAccess();
152     //Initial values for the GPIOS will be set on the lockdown version
153     ROVIM_T2D_ConfigDefaultParamBlock();
154
155     DEBUG_MSG("ROVIM T2D initialization complete.\r\n");
156     return;
157 }
158
159 void ROVIM_T2D_LockUnusedResourcesAccess(void)
160 {
161     ERROR_MSG("LockUnusedResourcesAccess not implemented. Don't worry about it.\r\n");
162 }
163
164 void ROVIM_T2D_ConfigSerialPort(void)
165 {
166     BYTE nBR=0;
167     /*If the serial port isn't configured with the correct baud rate, configure it now and
        reboot, to
168     force new configuration. This is only needed when the default parameter block is
        restored.*/
169     nBR = ReadExtEE_Byte(0x006D);
170     if(nBR != ROVIM_T2D_NBR)
171     {
172         WARNING_MSG("Configuring dalf baud rate parameter, nBR, to %d (see dalf owner's
            manual \
173 for details). If you're reading this, you have to reconfigure the terminal emulator to the
    new \
174 baud rate.\r\n", ROVIM_T2D_NBR);
175         WriteExtEE_Byte(0x006D,ROVIM_T2D_NBR);
176         Reset();
177     }
178     STATUS_MSG("Dalf baud rate parameter set to %d. If you can read this properly, you don't
         \
179 need to worry about it nor the gibberish printed above, if any.\r\n",ROVIM_T2D_NBR);
180 }
181
182 void ROVIM_T2D_ConfigDefaultParamBlock(void)
183 {
184     ROVIM_T2D_ConfigDirParamBlock();
185
186     //Traction encoder set up
187     //set TPR1 LSB
188     DEBUG_MSG("Setting TPR1 to %d ticks/rev. LSB=0x%02X, MSB=0x%02X.\r\n",
        ROVIM_T2D_TRACTION_TPR, ROVIM_T2D_TRACTION_TPR, 0);
189     CMD='W';
190     ARGN=4;
191     ARG[0]=1;
192     ARG[1]=01;
193     ARG[2]=0x40;
194     ARG[3]=ROVIM_T2D_TRACTION_TPR;
195     TeCmdDispatchExt();
196     //set TPR1 MSB
197     CMD='W';
198     ARGN=4;
```

```
199     ARG[0]=1;
200     ARG[1]=01;
201     ARG[2]=0x41;
202     ARG[3]=0;
203     TeCmdDispatchExt();
204     //set VSP1
205     DEBUG_MSG("Setting VSP1 to %d ms.\r\n",ROVIM_T2D_VSP1);
206     CMD='W';
207     ARGN=4;
208     ARG[0]=1;
209     ARG[1]=01;
210     ARG[2]=0x37;
211     ARG[3]=ROVIM_T2D_VSP1;
212     TeCmdDispatchExt();
213     //TODO: VBCAL
214     //set VBWARN LSB
215     DEBUG_MSG("Setting VBWARN to %d mV. LSB=0x%02X, MSB=0x%02X.\r\n",ROVIM_T2D_VBWARN, (BYTE
            ) (ROVIM_T2D_VBWARN>>8), (BYTE) (ROVIM_T2D_VBWARN & 0xFF));
216     CMD='W';
217     ARGN=4;
218     ARG[0]=1;
219     ARG[1]=01;
220     ARG[2]=0x29;
221     ARG[3]=(BYTE) (ROVIM_T2D_VBWARN>>8);
222     TeCmdDispatchExt();
223     //set VBWARN MSB
224     CMD='W';
225     ARGN=4;
226     ARG[0]=1;
227     ARG[1]=01;
228     ARG[2]=0x2A;
229     ARG[3]=(BYTE) (ROVIM_T2D_VBWARN & 0xFF);
230     TeCmdDispatchExt();
231
232     /*//XXX: temp.
233     WARNING_MSG("For debug purposes, setting VBWARN to a low value. Remove when you're
            finished debugging\r\n");
234     CMD='W';
235     ARGN=4;
236     ARG[0]=1;
237     ARG[1]=01;
238     ARG[2]=0x29;
239     ARG[3]=0;
240     TeCmdDispatchExt();
241     //set VBWARN MSB
242     CMD='W';
243     ARGN=4;
244     ARG[0]=1;
245     ARG[1]=01;
246     ARG[2]=0x2A;
247     ARG[3]=0;
248     TeCmdDispatchExt();*/
249 }
250
251 void ROVIM_T2D_ConfigDirParamBlock(void)
252 {
253     //set MAXERR LSB
254     DEBUG_MSG("Setting MAXERR to 0x%02X. LSB=0x%02X, MSB=0x%02X.\r\n",
255     (ROVIM_T2D_DIR_TICK_UPPER_LIMIT-ROVIM_T2D_DIR_TICK_LOWER_LIMIT),
256     (ROVIM_T2D_DIR_TICK_UPPER_LIMIT-ROVIM_T2D_DIR_TICK_LOWER_LIMIT), 0);
257     CMD='W';
258     ARGN=4;
259     ARG[0]=1;
260     ARG[1]=01;
261     ARG[2]=0x2C;
262     ARG[3]=(ROVIM_T2D_DIR_TICK_UPPER_LIMIT-ROVIM_T2D_DIR_TICK_LOWER_LIMIT);
263     TeCmdDispatchExt();
264     //set MAXERR MSB
265     CMD='W';
266     ARGN=4;
267     ARG[0]=1;
268     ARG[1]=01;
269     ARG[2]=0x2D;
```

```
270     ARG[3]=0;
271     TeCmdDispatchExt();
272     //set MTR2_MODE1
273     DEBUG_MSG("Setting MTR2_MODE1 to 0x%02X.\r\n",ROVIM_T2D_DIR_MODE1);
274     CMD='W';
275     ARGN=4;
276     ARG[0]=1;
277     ARG[1]=01;
278     ARG[2]=0x46;
279     ARG[3]=ROVIM_T2D_DIR_MODE1;
280     TeCmdDispatchExt();
281     //set MTR2_MODE2
282     DEBUG_MSG("Setting MTR2_MODE2 to 0x%02X.\r\n",ROVIM_T2D_DIR_MODE2);
283     CMD='W';
284     ARGN=4;
285     ARG[0]=1;
286     ARG[1]=01;
287     ARG[2]=0x47;
288     ARG[3]=ROVIM_T2D_DIR_MODE2;
289     TeCmdDispatchExt();
290     //set MTR2_MODE3
291     DEBUG_MSG("Setting MTR2_MODE3 to 0x%02X.\r\n",ROVIM_T2D_DIR_MODE3);
292     CMD='W';
293     ARGN=4;
294     ARG[0]=1;
295     ARG[1]=01;
296     ARG[2]=0x48;
297     ARG[3]=ROVIM_T2D_DIR_MODE3;
298     TeCmdDispatchExt();
299     //set ACC2 LSB
300     DEBUG_MSG("Setting ACC2 to %d ticks/VSP2^2. LSB=0x%02X, MSB=0x%02X.\r\n",ACC2, (BYTE) (
            ACC2>>8), (BYTE) (ACC2 & 0xFF));
301     CMD='W';
302     ARGN=4;
303     ARG[0]=1;
304     ARG[1]=01;
305     ARG[2]=0x49;
306     ARG[3]=(BYTE) (ACC2 & 0xFF);
307     TeCmdDispatchExt();
308     //set ACC2 MSB
309     CMD='W';
310     ARGN=4;
311     ARG[0]=1;
312     ARG[1]=01;
313     ARG[2]=0x4A;
314     ARG[3]=(BYTE) (ACC2>>8);
315     TeCmdDispatchExt();
316     //set VMID2 LSB
317     DEBUG_MSG("Setting VMID2 to %d ticks/VSP2. LSB=0x%02X, MSB=0x%02X.\r\n",VMID2, (BYTE) (
            VMID2>>8), (BYTE) (VMID2 & 0xFF));
318     CMD='W';
319     ARGN=4;
320     ARG[0]=1;
321     ARG[1]=01;
322     ARG[2]=0x4B;
323     ARG[3]=(BYTE) (VMID2 & 0xFF);
324     TeCmdDispatchExt();
325     //set VMID2 MSB
326     CMD='W';
327     ARGN=4;
328     ARG[0]=1;
329     ARG[1]=01;
330     ARG[2]=0x4C ;
331     ARG[3]=(BYTE) (VMID2>>8);
332     TeCmdDispatchExt();
333     //set VSP2
334     DEBUG_MSG("Setting VSP2 to %d ms.\r\n",ROVIM_T2D_DIR_VSP);
335     CMD='W';
336     ARGN=4;
337     ARG[0]=1;
338     ARG[1]=01;
339     ARG[2]=0x4D;
340     ARG[3]=ROVIM_T2D_DIR_VSP;
```

```
341     TeCmdDispatchExt();
342     //set VMIN2
343     DEBUG_MSG("Setting VMIN2 to %d %%.\r\n",ROVIM_T2D_DIR_MIN_PWM);
344     CMD='W';
345     ARGN=4;
346     ARG[0]=1;
347     ARG[1]=01;
348     ARG[2]=0x54;
349     ARG[3]=ROVIM_T2D_DIR_MIN_PWM;
350     TeCmdDispatchExt();
351     //set VMAX2
352     DEBUG_MSG("Setting VMAX2 to %d %%.\r\n",ROVIM_T2D_DIR_MAX_PWM);
353     CMD='W';
354     ARGN=4;
355     ARG[0]=1;
356     ARG[1]=01;
357     ARG[2]=0x55;
358     ARG[3]=ROVIM_T2D_DIR_MAX_PWM;
359     TeCmdDispatchExt();
360     //set TPR2 LSB
361     DEBUG_MSG("Setting TPR2 to %d ticks/rev. LSB=0x%02X, MSB=0x%02X.\r\n",256, 0, 1);
362     CMD='W';
363     ARGN=4;
364     ARG[0]=1;
365     ARG[1]=01;
366     ARG[2]=0x56;
367     ARG[3]=0x00;     //According to Dalf OM, TRP must be set to 0x100 for analog encoders
368     TeCmdDispatchExt();
369     //set TPR2 MSB
370     CMD='W';
371     ARGN=4;
372     ARG[0]=1;
373     ARG[1]=01;
374     ARG[2]=0x57;
375     ARG[3]=0x01;     //According to Dalf OM, TRP must be set to 0x100 for analog encoders
376     TeCmdDispatchExt();
377     //set MIN2 LSB
378     DEBUG_MSG("Setting MIN2 to 0x%2X. LSB=0x%02X, MSB=0x%02X.\r\n",
            ROVIM_T2D_DIR_TICK_LOWER_LIMIT, ROVIM_T2D_DIR_TICK_LOWER_LIMIT, 0);
379     CMD='W';
380     ARGN=4;
381     ARG[0]=1;
382     ARG[1]=01;
383     ARG[2]=0x58;
384     ARG[3]=ROVIM_T2D_DIR_TICK_LOWER_LIMIT;
385     TeCmdDispatchExt();
386     //set MIN2 MSB
387     CMD='W';
388     ARGN=4;
389     ARG[0]=1;
390     ARG[1]=01;
391     ARG[2]=0x59;
392     ARG[3]=0;
393     TeCmdDispatchExt();
394     //set MAX2 LSB
395     DEBUG_MSG("Setting MAX2 to 0x%2X. LSB=0x%02X, MSB=0x%02X.\r\n",
            ROVIM_T2D_DIR_TICK_UPPER_LIMIT, ROVIM_T2D_DIR_TICK_UPPER_LIMIT, 0);
396     CMD='W';
397     ARGN=4;
398     ARG[0]=1;
399     ARG[1]=01;
400     ARG[2]=0x5A;
401     ARG[3]=ROVIM_T2D_DIR_TICK_UPPER_LIMIT;
402     TeCmdDispatchExt();
403     //set MAX2 MSB
404     CMD='W';
405     ARGN=4;
406     ARG[0]=1;
407     ARG[1]=01;
408     ARG[2]=0x5B;
409     ARG[3]=0;
410     TeCmdDispatchExt();
411     //set DMAX
```

```
412     DEBUG_MSG("Setting DMAX to 0x%02X.\r\n",ROVIM_T2D_DMAX);
413     CMD='W';
414     ARGN=4;
415     ARG[0]=1;
416     ARG[1]=01;
417     ARG[2]=0x75;
418     ARG[3]=ROVIM_T2D_DMAX;
419     TeCmdDispatchExt();
420     //set FENBL
421     DEBUG_MSG("Setting FENBL to 0x%02X.\r\n",ROVIM_T2D_FENBL);
422     CMD='W';
423     ARGN=4;
424     ARG[0]=1;
425     ARG[1]=01;
426     ARG[2]=0x76;
427     ARG[3]=ROVIM_T2D_FENBL;
428     TeCmdDispatchExt();
429     //set DECAY
430     DEBUG_MSG("Setting DECAY to 0x%02X [If ADC sample time params are default: Fc=-ln(DECAY
            /256)*220/2/pi].\r\n",ROVIM_T2D_DECAY);
431     CMD='W';
432     ARGN=4;
433     ARG[0]=1;
434     ARG[1]=01;
435     ARG[2]=0x77;
436     ARG[3]=ROVIM_T2D_DECAY;
437     TeCmdDispatchExt();
438 }
439
440 //Start all remaining ROVIM features.
441 void ROVIM_T2D_Start(void)
442 {
443     ROVIM_T2D_sysmonitorcount = ROVIM_T2D_SYSTEM_MONITOR_PERIOD;
444     ROVIM_T2D_pwmrefreshcount = ROVIM_T2D_PWM_REFRESH_PERIOD;
445 }
446
447 void ROVIM_T2D_ConfigGPIOs(void)
448 {
449     BYTE i=0;
450     IOPinConfig config={0};
451
452     GPIOsDescription = DefaultGPIOsDescription;
453     ngpios= nDefaultgpios;
454
455     for (i=0; i<ngpios; i++) {
456         //copy configuration to data ram before calling setup function
457         if(strcmppgm("",GPIOsDescription[i].name)==0)
458         {
459             DEBUG_MSG("GPIO number %d unused.\r\n",GPIOsDescription[i].config.number);
460             continue;
461         }
462         memcpypgm2ram(&config,(const rom void *) &GPIOsDescription[i].config,sizeof(config))
                ;
463         SetGPIOConfig(GPIOsDescription[i].name,&config);
464     }
465
466     return;
467 }
468
469 void ROVIM_T2D_Greeting(void)
470 {
471     if(SCFG == TEcfg)
472     { // If Terminal Emulator Interface
473         Greeting();
474         printf("ROVIM T2D Brain\r\n");
475         printf("ROVIM T2D Software Ver:%2u.%u\r\n",ROVIM_T2D_SW_MAJOR_ID,
                ROVIM_T2D_SW_MINOR_ID);    // ROVIM Software ID
476         printf("ROVIM T2D Release date: "ROVIM_T2D_RELEASE_DATE"\r\n");
                                            // ROVIM Software release date
477         printf("ROVIM T2D Contact(s):\r\n"ROVIM_T2D_CONTACTS"\r\n");
                                              // ROVIM Contacts
478         printf("\r\n");
479     }
```

```
480 }
481
482 BYTE ROVIM_T2D_CmdDispatch(void)
483 {
484     if( ROVIM_T2D_IsCommandLocked(ARG[0]) )
485     {
486         return eDisable;
487     }
488     ResetPrintFlags=TRUE;
489     switch(ARG[0])
490     {
491         case ROVIM_T2D_LOCKDOWN_CMD_CODE:
492             return ROVIM_T2D_Lockdown();
493         case ROVIM_T2D_RELEASE_CMD_CODE:
494             return ROVIM_T2D_ReleaseFromLockdown();
495         case ROVIM_T2D_SOFTSTOP_CMD_CODE:
496             return ROVIM_T2D_SoftStop();
497         case ROVIM_T2D_CONTROL_GPIO_CMD_CODE:
498             return ROVIM_T2D_ControlGPIO();
499         case ROVIM_T2D_ACCELERATE_CMD_CODE:
500             return ROVIM_T2D_Accelerate();
501         case ROVIM_T2D_DECELERATE_CMD_CODE:
502             return ROVIM_T2D_Decelerate();
503         case ROVIM_T2D_SET_MOVEMENT_CMD_CODE:
504             return ROVIM_T2D_SetMovement();
505         case ROVIM_T2D_TURN_CMD_CODE:
506             return ROVIM_T2D_Turn();
507         case ROVIM_T2D_DEBUG_CTRL_CMD_CODE:
508             return ROVIM_T2D_DebugControl();
509         //Add more ROVIM commands here
510         default:
511             ERROR_MSG("Command does not exist.\r\n");
512             return eParseErr;
513     }
514     ResetPrintFlags=FALSE;
515     ERROR_MSG("Unexpected program execution.\r\n");
516     return eDisable;
517 }
518
519 void ROVIM_T2D_LockCriticalResourcesAccess(void)
520 {
521     ResourcesLockFlag=TRUE;
522     return;
523 }
524
525 void ROVIM_T2D_UnlockCriticalResourcesAccess(void)
526 {
527     ResourcesLockFlag=FALSE;
528     return;
529 }
530
531 BOOL ROVIM_T2D_IsCommandLocked(BYTE cmd)
532 {
533     BYTE i;
534
535     if(!ResourcesLockFlag)
536     {
537         return FALSE;
538     }
539
540     for(i=0;i<ROVIM_T2D_nCommandsToAllow;i++)
541     {
542         if( cmd==ROVIM_T2D_CommandsToAllow[i])
543         {
544             DEBUG_MSG("Found cmd=%d is in whitelist position %d.\r\n", cmd, i);
545             return FALSE;
546         }
547     }
548     ERROR_MSG("Command is locked.\r\n");
549     return TRUE;
550 }
551
552 BOOL ROVIM_T2D_FinishReleaseFromLockdown(void)
```

```
553 {
554     UnlockCriticalResourcesAccess();
555     //ROVIM_T2D_ConfigDefaultParamBlock(); //see if with soft stop this isn't needed
556     ResetGPIO("brake unclamper");
557     inLockdown=FALSE;
558     LedErr &= ~Lckmsk;
559     STATUS_MSG("ROVIM is now ready to move. Restart traction controller to clear any
            remaining \
560 error\r\n");
561
562     return TRUE;
563 }
564
565 BOOL ROVIM_T2D_LockBrake(void)
566 {
567     DEBUG_MSG("Locking emergency brake now.\r\n");
568     SetGPIO("brake clamper");
569     ResetGPIO("brake unclamper");
570
571     return TRUE;
572 }
573
574 BOOL ROVIM_T2D_UnlockBrake(void)
575 {
576     //Uncomment this message when braking is done automatically
577     //DEBUG_MSG("Unlocking brake now.\r\n");
578     ResetGPIO("brake clamper");
579     /*Depending on hw configuration, the unclamp pin may be unwired, so force this action to
            be done
580     manually. Regardless of that, the unlock procedure is the same*/
581     SetGPIO("brake unclamper");
582
583     return TRUE;
584 }
585
586 //my birthday's june 16th. I have an amazon wishlist. No pressure, though...
587 BOOL ROVIM_T2D_ValidateState(void)
588 {
589     BYTE directionError=0;
590     BYTE emergencyStop=0;
591     WORD delay=0;
592     TIME now;
593     //Since the system always goes to lockdown on power up, this initial value is accurate
            enough
594     static TIME before={0};
595     TIME EmergencyConditionTestStart;
596
597
598     /*I'm choosing not to monitor the brake clamp & unclamp & unclamp command here because:
599     the user may start unclamping before sending the command; while unclamping, there is
            nothing
600     I can do to stop that; after unclamping, if there is still a related (with these 3 GPIOs
            )
601     error condition, it will be picked up by the monitor task*/
602
603     GetGPIO("direction error switch",&directionError);
604     if(directionError)
605     {
606         ERROR_MSG("The direction end-of-travel switch is still ON.\r\n");
607         DEBUG_MSG("directionError=%d.\r\n",directionError);
608         return FALSE;
609     }
610     if(vel1)
611     {
612         ERROR_MSG("The traction is still moving.\r\n");
613         DEBUG_MSG("vel1=%ld.\r\n",vel1);
614         return FALSE;
615     }
616     if(Power2)
617     {
618         ERROR_MSG("The direction is still being powered.\r\n");
619         DEBUG_MSG("Power2=%d, (Mtr2_Flags1 & pida_Msk)=%d.\r\n", Power2, (Mtr2_Flags1 &
                pida_Msk));
```

```
620         return FALSE;
621     }
622
623     /*Make sure the brake clamper GPIO is off before testing the emergency stop condition*/
624     GetTime(&now);
625     delay=CalculateDelayMs(&before, &now);
626     if(ROVIM_T2D_BRAKE_CLAMP_TIME > delay)
627     {
628         ERROR_MSG("Brake clamper monostable is still ON. Do not forget that it activates
                every \
629 time you call this command. Wait %d ms before trying again.\r\n",
630         ROVIM_T2D_BRAKE_CLAMP_TIME);
631         DEBUG_MSG("delay=%d,timeout=%d.\r\n",delay, ROVIM_T2D_BRAKE_CLAMP_TIME);
632         return FALSE;
633     }
634     /*So, the software is too fast to bring down the brake clamper output and test the
            emergency condition
635     input sequentially. So we must wait for the output to go down. We do that with some busy
            waiting.
636     Because life's long enough to do some busy waiting */
637     GetTime(&before);
638     ResetGPIO("brake clamper");
639     SetDelay(&before, 0,msec_100);
640     while(!Timeout(&before));
641     GetGPIO("emergency stop condition",&emergencyStop);
642     /*We shouldn't decide here to release the brake, so we just put it back and start
            counting for the next time*/
643     SetGPIO("brake clamper");
644     GetTime(&before);
645
646     if(emergencyStop)
647     {
648         ERROR_MSG("There is still an emergency stop condition active.\r\n");
649         return FALSE;
650     }
651
652     DEBUG_MSG("Inputs are good. Make sure outputs are, too.\r\n");
653     //Stop motors
654     SoftStop(2);
655     SoftStop(3);
656
657     DEBUG_MSG("Vehicle is ready to be unclamped.\r\n");
658
659     return TRUE;
660 }
661
662
663 //-----------------------------------Periodic tasks of system monitoring
        -----------------------------------
664 void ROVIM_T2D_MonitorSystem(void)
665 {
666     BYTE previousVerbosity=0;
667     BYTE error=0;
668     BOOL finishUnlocking=FALSE;
669
670     TIME start={0}, stop={0};
671     static TIME prev={0};
672     DWORD delay=0;
673     static DWORD maxDelay=0;
674     GetTime(&start);
675
676     previousVerbosity = GetVerbosity();
677     if (!ManualSysMonitoring)    //in manual mode we do not need to restrict verbosity
678     {
679         /*We only want to let pass warning and errors. Status*/
680         SetVerbosity(VERBOSITY_LEVEL_ERROR | VERBOSITY_LEVEL_WARNING);
681     }
682
683     //Look for unrecoverable error conditions
684     if( ROVIM_T2D_DetectFatalError(ROVIM_T2D_SYSTEM_MONITOR_PERIOD))
685     {
686         Reset();
687     }
```

```
688
689     //Look for severe error conditions
690     error  = ROVIM_T2D_DetectTractionEror(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
691     error |= ROVIM_T2D_DetectBrakingError(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
692     error |= ROVIM_T2D_DetectDirectionEror(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
693     if(error)
694     {
695         ROVIM_T2D_Lockdown();
696     }
697     else
698     {
699         DEBUG_MSG("Yay, there is no need to go to lockdown.\r\n");
700     }
701
702     //look for and act on non-severe error conditions
703     ROVIM_T2D_MonitorTractionWarning(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
704     ROVIM_T2D_MonitorBrakingWarning(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
705     ROVIM_T2D_MonitorDirectionWarning(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
706
707     //ROVIM_T2D_MonitorEmergencyConditionInspection(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
708     //look for and act on brake unlock completion
709     finishUnlocking=ROVIM_T2D_DetectBrakeUnlock(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
710
711     //Detect and act on assynchronous switch to manual mode
712     ROVIM_T2D_MonitorManualMode(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
713
714     ROVIM_T2D_DetectSigmaDError(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
715
716     ROVIM_T2D_PendingCmd(ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
717
718     if (!ManualSysMonitoring)
719     {
720         SetVerbosity(previousVerbosity);
721     }
722
723     if(finishUnlocking)
724     {
725         //finished unlocking brake inside the safety timeout
726         ROVIM_T2D_FinishReleaseFromLockdown();
727         UnlockingBrake=FALSE;
728     }
729
730     if(ForcePrintMsg) ForcePrintMsg=FALSE;
731
732     //time measurements - debug purposes
733     GetTime(&stop);
734     delay=CalculateDelayMs(&start,&stop);
735     if(delay > maxDelay)
736     {
737         maxDelay=delay;
738         DEBUG_MSG("Monitor task max delay so far: %ld ms.\r\n",maxDelay);
739     }
740     //DEBUG_MSG("Monitor task delay: %ld ms.\r\n",delay);
741     delay=CalculateDelayMs(&prev,&start);
742     if(delay < ROVIM_T2D_SYSTEM_MONITOR_PERIOD)
743     {
744         DEBUG_MSG("Monitoring period smaller than expected. measured=%ld, expected=%ld.\r\n"
                , delay, ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
745     }
746     //DEBUG_MSG("Monitoring period measured=%ld, expected=%ld.\r\n", delay, (long)
            ROVIM_T2D_SYSTEM_MONITOR_PERIOD);
747     GetTime(&prev);
748
749     return;
750 }
751
752     ////////////////////////////////////////////////////////////////////
753     //      P E N D I N G    C M D    S E R V I C E S                  //
754     //                                                                //
755     //  If motor now stopped (Power1=0 or Power2=0), and pending      //
756     //  cmd is awaiting motor stopped, do it now.                     //
757     ////////////////////////////////////////////////////////////////////
758 void ROVIM_T2D_PendingCmd(BYTE period)
```

```
759 {
760     if ( (desiredMovement.type==FORWARD) && (!vel1)  && (movementType!=FORWARD) )
761     {
762         //pins are active low. Order of activation is important here.
763         SetGPIO("engage handbrake");
764         ResetGPIO("activate traction");
765         SetGPIO("engage reverse");
766         ResetGPIO("engage forward");
767         ROVIM_T2D_SetSpeed(desiredMovement.speed);
768         movementType=FORWARD;
769         STATUS_MSG("Vehicle moving forward at approximately %d km/h/10.\r\n",desiredMovement
                .speed);
770     }
771
772     if ( (desiredMovement.type==REVERSE) && (!vel1)  && (movementType!=REVERSE) )
773     {
774         //pins are active low. Order of activation is important here.
775         SetGPIO("engage handbrake");
776         ResetGPIO("activate traction");
777         SetGPIO("engage forward");
778         ResetGPIO("engage reverse");
779         ROVIM_T2D_SetSpeed(desiredMovement.speed);
780         movementType=REVERSE;
781         STATUS_MSG("Vehicle moving backwards at approximately %d km/h/10.\r\n",
                desiredMovement.speed);
782     }
783     DEBUG_MSG("desired type=%d, speed=%d, current type=%d, vel1=%ld, FW=%d,RV=%d.\r\n",
784     desiredMovement.type, desiredMovement.speed, movementType, vel1, FORWARD,REVERSE);
785     //neutral mode can be done immediately. There's no need to wait until the vehicle stops.
786 }
787
788 void ROVIM_T2D_DetectSigmaDError(BYTE period)
789 {
790     static long ledOffTime=0;
791     static BOOL printMsg=TRUE;
792     WORD led=0;
793
794
795     if(ForcePrintMsg) printMsg=TRUE;
796
797     led=AdcConvert((WORD)SigmaDLed);
798
799     /*FYI: Led Off: V ~=4900 mV; Led ON: V~=2300 mV.*/
800     if(led > 4500) //this threshold != 5V is to avoid false positives (we are reading
            analogue)
801     {
802         ledOffTime+=(long)period;
803     }
804     else
805     {
806         ledOffTime=0;
807         SigmaDError=TRUE;
808         if(printMsg)
809         {
810             WARNING_MSG("The SigmaDrive traction controller either is turned off or has
                    encountered an error.\r\n");
811             printMsg=FALSE;
812         }
813     }
814
815     if(ledOffTime >= (long)ROVIM_T2D_SIGMAD_ERROR_TIMEOUT)
816     {
817         SigmaDError=FALSE;
818         printMsg=TRUE;
819     }
820 }
821
822 void ROVIM_T2D_MonitorManualMode(BYTE period)
823 {
824     BYTE autoSwitch=0;
825     BYTE dcc=0, acc=0, fw=0, rv=0, handbrake=0;
826
827     GetGPIO("auto mode switch",&autoSwitch);
```

```
828     GetGPIO("decelerator",&dcc);
829     GetGPIO("accelerator",&acc);
830     GetGPIO("engage forward",&fw);
831     GetGPIO("engage reverse",&rv);
832     GetGPIO("engage handbrake",&handbrake);
833
834     if(!autoSwitch)
835     {
836         /*Make sure the outputs are in a good state. If we don't do this, the vehicle may
                start
837         moving unexpectedly when the user switches to auto mode. We must keep monitoring
                these
838         variables because we must assume the user is stupid and will try to send commands
                while
839         on manual mode, and we have no interrupt associated with this pin.*/
840         if(Power2)
841         {
842             WARNING_MSG("Detected incoherent direction outputs because of manual mode.
                    Resetting them.\r\n");
843             SoftStop(2);
844         }
845         if((dcc) || (acc) || (!fw) || (!rv) || (!handbrake))
846         {
847             WARNING_MSG("Detected incoherent traction outputs because of manual mode.
                    Resetting them.\r\n");
848             SoftStop(3);
849         }
850         autoMode=FALSE;
851     }
852     else
853     {
854         autoMode=TRUE;
855         DEBUG_MSG("Detected automatic mode of operation.\r\n");
856     }
857 }
858
859 //detects when traction system has encountered a serious error. It is in a Not OK state
860 BOOL ROVIM_T2D_DetectTractionEror(BYTE period)
861 {
862     static WORD movingOnHoldTimeout=0;
863     static BOOL printMsg=TRUE;
864
865     if(ForcePrintMsg) printMsg=TRUE;
866
867     /*XXX: Not ready yet. Must be done only with closed loop speed control
868     if ((desiredMovement.speed < vel1) && (settlingTime > ROVIM_T2D_MAX_SETTLING_TIME))
869     {
870         ERROR_MSG("Vehicle is taking too long to reduce its speed.\r\n");
871         DEBUG_MSG("Desired vel=%d, vel=%ld, settlingTime=%d.\r\n", desiredMovement.speed,
                vel1, settlingTime);
872         return TRUE;
873     }*/
874     if (vel1 > ROVIM_T2D_CRITICAL_SPEED)
875     {
876         if(printMsg)
877         {
878             ERROR_MSG("Vehicle is moving too fast.\r\n");
879             printMsg=FALSE;
880         }
881         DEBUG_MSG("vel=%ld, max speed=%ld.\r\n", vel1, ROVIM_T2D_CRITICAL_SPEED);
882         return TRUE;
883     }
884     /* We cannot get accurate enough acceleration readings from the speed encoder we have
            and it's current mounting to do this check
885     if ((abs(acc1)) > ROVIM_T2D_CRASH_ACC_THRESHOLD)
886     {
887         if(printMsg)
888         {
889             ERROR_MSG("Vehicle is accelerating too fast.\r\n");
890             printMsg=FALSE;
891         }
892         DEBUG_MSG("acc=%ld, max acc=%d.\r\n", acc1, ROVIM_T2D_CRASH_ACC_THRESHOLD);
893         return TRUE;
```

```
894     }*/
895     if ((desiredMovement.type==HILLHOLD) && (vel1) && (autoMode))
896     {
897         movingOnHoldTimeout+=period;
898     }
899     else
900     {
901         movingOnHoldTimeout=0;
902     }
903     if (movingOnHoldTimeout>ROVIM_T2D_MOVING_ON_HOLD_TIMEOUT)
904     {
905         if(printMsg)
906         {
907             ERROR_MSG("Vehicle should be on hold, yet it is moving.\r\n");
908             printMsg=FALSE;
909         }
910         DEBUG_MSG("movement type=%d, vel=%ld, timeout=%d.\r\n", desiredMovement.type, vel1,
                ROVIM_T2D_MOVING_ON_HOLD_TIMEOUT);
911         return TRUE;
912     }
913
914     printMsg=TRUE;
915     DEBUG_MSG("No traction errors detected.\r\n");
916     return FALSE;
917 }
918
919 //detects when braking system has encountered a serious error. It is in a Not OK state
920 BOOL ROVIM_T2D_DetectBrakingError(BYTE period)
921 {
922     BYTE endOfTravelClamp=0, endOfTravelUnclamp=0, unclampAction=0;
923     BYTE emergencyStop=0;
924     static BOOL printMsg=TRUE;
925
926     if(ForcePrintMsg) printMsg=TRUE;
927
928     GetGPIO("brake clamp switch",&endOfTravelClamp);
929     GetGPIO("brake unclamp switch",&endOfTravelUnclamp);
930     GetGPIO("emergency stop condition",&emergencyStop);
931     GetGPIO("brake unclamper",&unclampAction);
932
933     if((emergencyStop) && (!UnlockingBrake))
934     {
935         //detected emergency condition - Error!
936         if(printMsg)
937         {
938             ERROR_MSG("There is an emergency stop condition.\r\n");
939             printMsg=FALSE;
940         }
941         DEBUG_MSG("emergency command=%d. Remember this will occur while you have the 'brake
                clamper' ON.\r\n", emergencyStop);
942         return TRUE;
943     }
944     if((!endOfTravelClamp) && (!endOfTravelUnclamp) && (!unclampAction))
945     {
946         if(printMsg)
947         {
948             ERROR_MSG("Emergency brake is neither locked nor unlocked.\r\n");
949             printMsg=FALSE;
950         }
951         DEBUG_MSG("clamp end-of-travel=%d, unclamp end-of-travel=%d, unclamping=%d.\r\n",
                endOfTravelClamp, endOfTravelUnclamp,unclampAction);
952         return TRUE;
953     }
954
955     printMsg=TRUE;
956     DEBUG_MSG("No braking errors detected.\r\n");
957     return FALSE;
958 }
959
960 //detects when direction system has encountered a serious error. It is in a Not OK state
961 BOOL ROVIM_T2D_DetectDirectionEror(BYTE period)
962 {
963     BYTE endOfTravel=0;
```

```
964     static BOOL printMsg=TRUE;
965
966     if(ForcePrintMsg) printMsg=TRUE;
967
968     /* This check only makes sense if we can cut power to the direction motor when going to
            lockdown.
969     Currently we can't do so, so this stays commented.
970     pos=DirPos;
971     if ((pos > ROVIM_T2D_DIRECTION_CRITICAL_UPPER_POSITION) ||
972         (pos < ROVIM_T2D_DIRECTION_CRITICAL_LOWER_POSITION))
973     {
974         //reached soft direction position limit - Error!
975         return TRUE;
976     }*/
977
978     GetGPIO("direction error switch",&endOfTravel);
979     if (endOfTravel)
980     {
981         /*This should only be relevant in auto mode, but since this condition is hardwired
                to
982         force the vehicle to go to lockdown, we let the software follow along*/
983         //reached hard direction position limit - Error!
984         if(printMsg)
985         {
986             ERROR_MSG("Direction has reached one end-of-travel switch.\r\n");
987             printMsg=FALSE;
988         }
989         DEBUG_MSG("Switch=%d.\r\n", endOfTravel);
990         return TRUE;
991     }
992
993     printMsg=TRUE;
994     DEBUG_MSG("No direction errors detected.\r\n");
995     return FALSE;
996 }
997
998 void ROVIM_T2D_MonitorTractionWarning(BYTE period)
999 {
1000     //DEBUG_MSG("No traction warnings detected.\r\n");
1001 }
1002
1003 void ROVIM_T2D_MonitorBrakingWarning(BYTE period)
1004 {
1005     //DEBUG_MSG("No braking warnings detected.\r\n");
1006 }
1007
1008 void ROVIM_T2D_MonitorDirectionWarning(BYTE period)
1009 {
1010     /* Detect if the motor is under a big load for a long period of time.
1011     Since we do not have current sensors for the motors, we try to emulate that monitoring
            the
1012     PWM duty cycle. If it stays for too long too high, it means the system is not being able
            to
1013     reach a stable state, and the behaviour is in some sort erroneous.
1014     We chose the absolute number (since power up) of stressed "samples" as a metric instead
            of
1015     the relative (since last move command) because it reduces complexity (communication
            between
1016     functions is simplified). Also we believe in the long term this count should accuratelly
            reflect
1017     the amount of stress the motor is having.
1018     NOTE: We should be doing this check every at VSP intervals...*/
1019     /*
1020     //Only detect the stress if the motor if PID is active
1021     if (Mtr2_Flags1 & pida_Msk)
1022     {
1023         // Due to the nature of PID control, the motor may not be at full power and still be
                in stress
1024         if (Power2 > (VMAX1/ROVIM_T2D_DIRECTION_MOTOR_STRESS_DUTY_CYCLE_THRESHOLD))
1025         {
1026             MotorStressMonitor[2]++;
1027         }
1028         if (MotorStressMonitor[2] > ROVIM_T2D_DIRECTION_MOTOR_STRESS_CNT_THRESHOLD)
```

```
1029            {
1030                //reached direction motor overstress condition – Error!
1031                ERROR_MSG("ROVIM direction motor is too stressed. Going to stop it for
                        precaution.\r\n");
1032                DEBUG_MSG("MotorStressMonitor[2]=%d, threshold # stress events=%d.\r\n",
1033                MotorStressMonitor[2], ROVIM_T2D_DIRECTION_MOTOR_STRESS_CNT_THRESHOLD);
1034                return;
1035            }
1036        }*/
1037        //DEBUG_MSG("No direction warnings detected.\r\n");
1038    }
1039
1040    //detects when the system has encountered an error from which it cannot recover
1041    BOOL ROVIM_T2D_DetectFatalError(WORD period)
1042    {
1043        static WORD unclampActionActiveTime=0;
1044        BYTE unclampAction=0;
1045        BYTE endOfTravelClamp=0;
1046        BYTE emergencyStop=0;
1047
1048        /*TODO:
1049        GetGPIO("brake clamp switch",&endOfTravelClamp);
1050        GetGPIO("emergency stop condition",&emergencyStop);
1051        GetGPIO("brake unclamp command",&unclampAction);
1052
1053        if(unclampAction)
1054        {
1055            unclampActionActiveTime+=period;
1056        }
1057        else
1058        {
1059            unclampActionActiveTime=0;
1060        }
1061        if(unclampActionActiveTime > ROVIM_T2D_FATAL_UNCLAMP_TIMEOUT)
1062        {
1063            FATAL_ERROR_MSG("User is pressing the button for too long – or something else really
                        , really \
1064    bad happened.\r\n");
1065            DEBUG_MSG("Timeout=%d, time passed=%d.\r\n",ROVIM_T2D_FATAL_UNCLAMP_TIMEOUT,
                        unclampActionActiveTime));
1066            return TRUE;
1067        }*/
1068
1069        /* Need to have a timeout here, because of the inertia
1070        GetGPIO("traction voltage sensor", &tractionVoltage);
1071        if ((!tractionVoltage) && (vel1))
1072        {
1073            //This means either the contactor, fuse or battery for the traction system is NOK.
                        The vehicle is not in neutral
1074            FATAL_ERROR_MSG("Vehicle is moving, yet is not being powered.");
1075            DEBUG_MSG("VB+=%d, vel=%ld.\r\n", tractionVoltage, vel1);
1076            return TRUE;
1077        }*/
1078        DEBUG_MSG("No fatal errors detected.\r\n");
1079        return FALSE;
1080    }
1081
1082    #if 0
1083    BOOL ROVIM_T2D_MonitorEmergencyConditionInspection(WORD period)
1084    {
1085        static WORD timeoutCount=0;
1086        BYTE emergencyStop=0;
1087
1088        if(!InspectingEmergencyCondition)
1089        {
1090            timeoutCount=0;
1091            return FALSE;
1092        }
1093
1094        timeoutCount+=period;
1095        /*Before we can test the brake clamper switch, we must make sure the emergency stop
                    condition is off.
```

```
1096      But what we really want to know is exactly that, so we test it directly instead of the
              switch*/
1097      GetGPIO("emergency stop condition",&emergencyStop);
1098      if ((!emergencyStop) && (timeoutCount <= ROVIM_T2D_TEST_EMERGENCY_CONDITION_TIMEOUT))
1099      {
1100          //this must be done outside this function so that the status msg gets printed
1101          return TRUE;
1102      }
1103      if (timeoutCount > ROVIM_T2D_TEST_EMERGENCY_CONDITION_TIMEOUT)
1104      {
1105          //timeout expired. Going back to full lockdown
1106          SetGPIO("brake clamper");
1107          ResetGPIO("brake unclamper");
1108          ERROR_MSG("Took too long to test for an emergency stop condition. try again\r\n");
1109          DEBUG_MSG("stop condition=%d, time passed=%d > timeout=%d, inspecting emergency stop
                  =%d.\r\n",emergencyStop, timeoutCount,
                  ROVIM_T2D_TEST_EMERGENCY_CONDITION_TIMEOUT, InspectingEmergencyCondition);
1110          InspectingEmergencyCondition=FALSE;
1111      }
1112      return FALSE;
1113
1114
1115
1116      GetGPIO("emergency stop condition",&emergencyStop);
1117      /*We shouldn't decide here to release the brake, so we just put it back and start
              counting for the next time*/
1118      SetGPIO("brake clamper");
1119      GetTime(&before);
1120
1121      if(emergencyStop)
1122      {
1123          ERROR_MSG("There is still an emergency stop condition active.\r\n");
1124          return FALSE;
1125      }
1126
1127      DEBUG_MSG("Inputs are good. Make sure outputs are, too.\r\n");
1128      //Stop motors
1129      SoftStop(2);
1130      SoftStop(3);
1131
1132      DEBUG_MSG("Vehicle is ready to be unclamped.\r\n");
1133  }
1134  #endif
1135
1136  BOOL ROVIM_T2D_DetectBrakeUnlock(WORD period)
1137  {
1138      static WORD timeoutCount=0;
1139      BYTE endOfTravelUnclamp=0;
1140
1141      if(!UnlockingBrake)
1142      {
1143          timeoutCount=0;
1144          return FALSE;
1145      }
1146
1147      timeoutCount+=period;
1148      GetGPIO("brake unclamp switch",&endOfTravelUnclamp);
1149      if ((endOfTravelUnclamp) && (timeoutCount <= ROVIM_T2D_BRAKE_UNLOCK_TIMEOUT))
1150      {
1151          //this must be done outside this function so that the status msg gets printed
1152          return TRUE;
1153      }
1154      if (timeoutCount > ROVIM_T2D_BRAKE_UNLOCK_TIMEOUT)
1155      {
1156          //timeout expired. Going back to full lockdown
1157          SetGPIO("brake clamper");
1158          ResetGPIO("brake unclamper");
1159          ERROR_MSG("Brake unlocking took too long. Restart the procedure.\r\n");
1160          DEBUG_MSG("brake unlock end-of-travel=%d, time passed=%d > timeout=%d, unlocking=%d
                  .\r\n",endOfTravelUnclamp, timeoutCount, ROVIM_T2D_BRAKE_UNLOCK_TIMEOUT,
                  UnlockingBrake);
1161          UnlockingBrake=FALSE;
1162      }
```

```
1163        return FALSE;
1164    }
1165
1166
1167    //
            ------------------------------------------------------------------------------


1168    //Maintain PWM signal used by ROVIM T2D motors controlled by GPIO
1169    //motors controller by the Dalf defaullt motor interface are outside the scope of this
            function
1170    void ROVIM_T2D_ServicePWM(void)
1171    {
1172        static BYTE accDutyCtrl=0, dccDutyCtrl=0;
1173        static WORD iDebug=1, jDebug=1; //debug stuff
1174        static BYTE printctrl=0, dccprintctrl=0; //debug stuff
1175        BYTE previousVerbosity=0;
1176        TIME start={0}, stop={0};
1177        DWORD delay=0;
1178        static DWORD maxDelay=0;
1179        GetTime(&start);
1180
1181        previousVerbosity = GetVerbosity();
1182        SetVerbosity(VERBOSITY_LEVEL_ERROR | VERBOSITY_LEVEL_WARNING);
1183        /*DebugPWM and printctrl mechanism:
1184        DebugPWM is the global debug enable. OFF= normal operation. ON= debug mode
1185        printctrl controls the prints, to avoid too much info. Only on the first time a new if
                case is
1186        entered is debug info printed. This flag is reset on the next if case entered.
1187        dccprintctrl is like printctrl, but for dcc only. This is needed because Acc and Dcc ifs
                are not
1188        mutually exclusive.
1189        DebugPWM is also a counter controlling the amount of time debug is enabled.*/
1190        if(DebugPWM)
1191        {
1192            //debug stuff
1193            if (!printctrl)
1194            {
1195                printctrl=0xFF;
1196            }
1197            if(!dccprintctrl)
1198            {
1199                dccprintctrl=0xFF;
1200            }
1201        }
1202        else
1203        {
1204            //debug stuff
1205            printctrl=0;
1206            dccprintctrl=0;
1207        }
1208        /*DEBUG_MSG("Running PWM refresh thread. accDutyCtrl=%d, dccDutyCtrl=%d, PeriodCnt=%d, \
1209    movementType=%d, WaitForAccDccDecay=%d, AccDutyCycle=%d, DccDutyCycle=%d.\r\n", accDutyCtrl,
1210    dccDutyCtrl, PeriodCnt, movementType, WaitForAccDccDecay, AccDutyCycle, DccDutyCycle); //
            debug stuff*/
1211
1212        /*
1213        //debug stuff
1214        //In order to not overload the serial with prints, we do some mambo-jambo here
1215        if(jDebug>100)iDebug=1;
1216        if(jDebug>10000){jDebug=1; DebugPWM--;}
1217        if(DebugPWM)
1218        {
1219            jDebug++;
1220            iDebug++;
1221        }*/
1222
1223        if(movementType==HILLHOLD)
1224        {
1225            //if(!(iDebug%4))MSG("On HILLHOLD.\r\n"); //debug stuff
1226            if ((DebugPWM) && (printctrl & 0x01))
1227            {
1228                //debug stuff
1229                MSG("Vehicle on hill hold. type=%d.\r\n",movementType);
```

```
1230                printctrl=0xFE;
1231            }
1232            if(DebugPWM) DebugPWM--;//otherwise this if case will hang forever with no new
                    information
1233            //make sure we have the signals down
1234            ResetGPIO("decelerator");
1235            ResetGPIO("accelerator");
1236            goto exit;
1237        }
1238        if(!autoMode)
1239        {
1240            //if(!(iDebug%4))MSG("On manual.\r\n"); //debug stuff
1241            if ((DebugPWM) && (printctrl & 0x02))
1242            {
1243                //debug stuff
1244                MSG("Vehicle on manual. type=%d.\r\n",autoMode);
1245                printctrl=0xFD;
1246            }
1247            if(DebugPWM) DebugPWM--;//otherwise this if case will hang forever with no new
                    information
1248            //make sure we have the signals down
1249            ResetGPIO("decelerator");
1250            ResetGPIO("accelerator");
1251            goto exit;
1252        }
1253        if(WaitForAccDccDecay!=0)
1254        {
1255            /* We should not have both signals active at the same time. We wait until the
                    previous one
1256            goes to 0 (or close to it), to activate the new one*/
1257            //if(!(iDebug%4))MSG("Decay!=0,=%d.\r\n",WaitForAccDccDecay); //debug stuff
1258            if ((DebugPWM) && (printctrl & 0x04))
1259            {
1260                //debug stuff
1261                MSG("Waiting for Acc/Dcc signal to fall before rising the other. \
1262 WaitForAccDccDecay=%d.\r\n", WaitForAccDccDecay);
1263                printctrl=0xFB;
1264            }
1265            WaitForAccDccDecay--;
1266            ResetGPIO("decelerator");
1267            ResetGPIO("accelerator");
1268
1269            goto exit;
1270        }
1271        if(PeriodCnt >= 100)
1272        {
1273            //if(!(iDebug%4))MSG("PeriodCnt=%d, >= 100.\r\n",PeriodCnt); //debug stuff
1274            if ((DebugPWM) && (printctrl & 0x08))
1275            {
1276                //debug stuff
1277                MSG("Resetting period counter. PeriodCnt=%d.\r\n",PeriodCnt);
1278                printctrl=0xF7;
1279                DebugPWM--;
1280            }
1281            accDutyCtrl=0;
1282            dccDutyCtrl=0;
1283            PeriodCnt=0;
1284        }
1285        PeriodCnt++;
1286        //accelerator pwm
1287        if(AccDutyCycle <= accDutyCtrl)
1288        {
1289            //if(!(iDebug%4))MSG("AccDutyCycle<=Ctrl, Ctrl=%d, Duty=%d.\r\n",accDutyCtrl,
                    AccDutyCycle); //debug stuff
1290            if ((DebugPWM) &&(printctrl & 0x10))
1291            {
1292                //debug stuff
1293                MSG("Acc=0 now. AccDutyCycle=%d, accDutyCtrl=%d, PeriodCnt=%d.\r\n",
                        AccDutyCycle, accDutyCtrl, PeriodCnt);
1294                printctrl=0xEF;
1295            }
1296            ResetGPIO("accelerator");
1297        }
```

```
1298        else
1299        {
1300            //if(!(iDebug%4))MSG("AccDutyCycle>Ctrl, Ctrl=%d, Duty=%d.\r\n", accDutyCtrl,
                AccDutyCycle); //debug stuff
1301            if ((DebugPWM) && (printctrl & 0x20))
1302            {
1303                //debug stuff
1304                MSG("Acc=1 now. AccDutyCycle=%d, accDutyCtrl=%d, PeriodCnt=%d.\r\n",
                    AccDutyCycle, accDutyCtrl, PeriodCnt);
1305                printctrl=0xDF;
1306            }
1307            SetGPIO("accelerator");
1308            accDutyCtrl++;
1309        }
1310        //decelerator pwm
1311        if(DccDutyCycle <= dccDutyCtrl)
1312        {
1313            //if(!(iDebug%4))MSG("DccDutyCycle<=Ctrl, Ctrl=%d, Duty=%d.\r\n", dccDutyCtrl,
                DccDutyCycle); //debug stuff
1314            if ((DebugPWM) && (dccprintctrl & 0x01))
1315            {
1316                //debug stuff
1317                MSG("Dcc=0 now. DccDutyCycle=%d, dccDutyCtrl=%d, PeriodCnt=%d.\r\n",
                    DccDutyCycle, dccDutyCtrl, PeriodCnt);
1318                dccprintctrl=0xFE;
1319            }
1320            ResetGPIO("decelerator");
1321        }
1322        else
1323        {
1324            //if(!(iDebug%4))MSG("DccDutyCycle>Ctrl, Ctrl=%d, Duty=%d.\r\n", dccDutyCtrl,
                DccDutyCycle); //debug stuff
1325            if ((DebugPWM) && (dccprintctrl & 0x02))
1326            {
1327                //debug stuff
1328                MSG("Dcc=1 now. DccDutyCycle=%d, dccDutyCtrl=%d, PeriodCnt=%d.\r\n",
                    DccDutyCycle, dccDutyCtrl, PeriodCnt);
1329                dccprintctrl=0xFD;
1330            }
1331            SetGPIO("decelerator");
1332            dccDutyCtrl++;
1333        }
1334
1335        /*We use a label here because there are many exit points and the exit code is quite
            large*/
1336    exit:
1337        SetVerbosity(previousVerbosity);
1338        GetTime(&stop);
1339        delay=CalculateDelayMs(&start,&stop);
1340        if(delay > maxDelay)
1341        {
1342            maxDelay=delay;
1343            DEBUG_MSG("PWM refresh task max delay so far: %d ms.\r\n",maxDelay);
1344        }
1345    }
1346
1347    void ROVIM_T2D_UpdateVelocity1(void)
1348    {
1349        long temp1=0, temp2=0, velRPM=0;
1350        long currVel=0;
1351        static long n1Vel=0, n2Vel=0, n3Vel=0;
1352        static long sumE1=0;
1353        static long t=0;
1354        short long E1=0;
1355        static long diffE1=0;    //tick count during VSP. Same as V1, but V1 seems much more
            inaccurate, so we use this
1356        static BYTE debugcnt=0, end=1; //debug stuff
1357        /*static long t2=0;
1358        static long sumE12=0;*/
1359
1360
1361        UpdateVelocity1();
1362
```

```
1363    INTCONbits.GIEH = 0;     // Disable high priority interrupts
1364    E1=encode1;
1365    INTCONbits.GIEH = 1;     // Enable high priority interrupts
1366
1367    diffE1=E1-diffE1;
1368    if(diffE1<0)
1369        diffE1=0;
1370
1371    sumE1+=(long)diffE1;
1372    if(sumE1<0)
1373        sumE1=0;
1374
1375    t+=(long)VSP1;
1376    diffE1=E1;
1377
1378    /* Velocity calculation scheme: We need enough samples to produce an accurate enough
            velocity reading.
1379    If the vehicle is moving too slowly, we just won't get them. So I say to only calculate
            the speed
1380    if we have a minimum number of new samples collected, or the timeout to get them has
            expired.
1381    Still, we may get sudden speed readings changes, that are not a representation of the
            physical
1382    process, i.e. the vehícle is moving relativelly steady. This may be due to several
            factors,
1383    such as the accelerator DAC or chain slack. So, to the actual speed acessible to the
            rest of the
1384    software is filtered through digital recursive filter with a decay parameter, just like
            the
1385    stock dalf firmware does for the ADC readings.
1386    The properties of the encoder sensor used and its mounting make the calculation of the
1387    acceleration pretty useless, since it produces even more inaccurate readings than the
            speed. So
1388    we are not going to use it.
1389    */
1390    if( (sumE1 >= ROVIM_T2D_VEL1_CALC_MIN_TICK_CNT) || (t >= (6*(long)VSP1)) )
1391    {
1392        temp1 = sumE1 * (long)60000 ;
1393        temp2 = t * TPR1 ;
1394        velRPM = temp1/temp2;    //I don't feeel like fixing the rounding mistake
1395        //vel (Km/10/h) = vel (RPM) * Perimeter (cm) * 3.6 /60 (s/min) / 100 (m/cm) / 10
1396        temp1 = velRPM * ROVIM_T2D_WHEEL_PERIMETER * 36;
1397        temp2 = 6000;
1398        currVel=temp1/temp2;    //I don't feeel like fixing the rounding mistake
1399
1400
1401        vel1=(currVel<<3) + (n1Vel<<2) + (n2Vel<<1) + (n3Vel<<1);
1402        vel1=vel1>>4;
1403        n3Vel=n2Vel;
1404        n2Vel=n1Vel;
1405        n1Vel = currVel;
1406
1407        /* acceleration not used - for now
1408        //acc(Km/10/h/s) = dV(km/h/10) / dt(ms) * 1000(ms/s)
1409        temp1 = (vel1-n1Vel)*1000;
1410        temp2 = t;
1411        acc1 = temp1/temp2; //I don't feeel like fixing the rounding mistake*/
1412
1413        debugcnt++;
1414        if(debugcnt>=end)
1415        {
1416            debugcnt=0;
1417            DEBUG_MSG("E1=%ld, V1=%ld, t=%ld, sum=%ld velRPM=%ld, vel1=%ld, n1Vel=%ld,
                    currvel=%ld, n2Vel=%ld, n3Vel=%ld.\r\n",
1418            diffE1, (long)V1, t, sumE1, velRPM, vel1, n1Vel, currVel, n2Vel, n3Vel);
1419        }
1420
1421        sumE1=0;
1422        t=0;
1423    }
1424    /*t2+=(long)VSP1;
1425    if(t2==60000)
1426    {
```

```
1427            sumE12=(long)E1-sumE12;
1428            ERROR_MSG("ticks=%ld.\r\n",sumE12);
1429            sumE12=E1;
1430            t2=0;
1431        }*/
1432    }
1433
1434    void ROVIM_T2D_FullBrake(void)
1435    {
1436        DEBUG_MSG("Applying full brake.\r\n");
1437        CMD='G';
1438        ARG[0]=ROVIM_T2D_DECELERATE_CMD_CODE;
1439        ARG[1]=100;
1440        ARGN=2;
1441        TeCmdDispatchExt();
1442    }
1443
1444    void ROVIM_T2D_SetSpeed(BYTE speed)
1445    {
1446        BYTE dutyCycle=0;
1447        long temp=0;
1448
1449        /*temp=(long)speed>>1;
1450        if(temp>=SpeedToDutyCycleLen)
1451        dutyCycle=SpeedToDutyCycle[temp]; //get the duty cycle from the LUT
1452        temp=(long)dutyCyle*SpeedDCScaling/10; //Scale the conversion. This can be changed in
                real time
1453        dutyCycle=(BYTE) temp;
1454        if (dutyCycle > 100) dutyCycle=100;*/
1455        //XXX: fix this
1456
1457        temp=speed*573;
1458        temp=temp+27959;
1459        temp=temp/1000;
1460        if ((speed<ROVIM_T2D_LOWER_SPEED_LIMIT) || (temp<0))
1461        {
1462            temp=0;
1463        }
1464        if(temp>100)
1465        {
1466            temp=100;
1467        }
1468        dutyCycle=(BYTE) temp;
1469
1470        CMD='G';
1471        ARG[0]=ROVIM_T2D_ACCELERATE_CMD_CODE;
1472        ARG[1]=dutyCycle;
1473        ARGN=2;
1474        TeCmdDispatchExt();
1475    }
1476
1477    //just like in dalf, this function only indicates the PWM set by the board,
1478    //it doesn't translate actual voltage fed to the motor (it may be disconnected, for example)
1479    BYTE ROVIM_T2D_LightPWMLed(BYTE dutyCycle, BYTE direction)
1480    {
1481        if(dutyCycle>100)
1482        {
1483            WARNING_MSG("Duty cycle larger than 100 %%.\r\n");
1484            dutyCycle=100;
1485        }
1486        if(direction>REVERSE)
1487        {
1488            ERROR_MSG("Unexpected direction.\r\n");
1489            return eParmErr;
1490        }
1491        /*To control the PWM led, it seems we need to actualy use the dalf firmware to drive the
                PWM.
1492        Since it isn't connected, there's no problem.*/
1493        CMD='X';
1494        ARG[0]=1;
1495        ARG[1]=direction;
1496        ARG[2]=dutyCycle;
1497        ARGN=3;
```

```
1498      TeCmdDispatchExt();
1499  }
1500
1501  void      ROVIM_T2D_ServiceLED(void)          // Periodic LED service
1502  {
1503  ////////////////////////////////////////////////////////////////////////////
1504  //            patterns on LED1 and LED2 indicating motor status:          //
1505  //                                                                        //
1506  //  LED1                                                                  //
1507  //  OFF:          Motor is not good to go (maybe still be ON)             //
1508  //  ON:           Motor is good to go.                                    //
1509  //                                                                        //
1510  //  LED2                                                                  //
1511  //  OFF:          Power=0. No Power applied. ------------------ Off        //
1512  //  FAST BLINK:   Power>0, TGA active.  Power applied. ------- TGA Active //
1513  //  SLOW BLINK:   Power>0, TGA inactive, Power applied, V>0. - Open loop move. //
1514  //  ON:           Power>0, TGA inactive, Power applied, V=0. - Stalled.   //
1515  //                                                                        //
1516  //                   Three possible patterns on LED3                      //
1517  //  OFF:           No Error.                                              //
1518  //  FAST BLINK:    ROVIM is in lockdown mode                              //
1519  //  SLOW BLINK:    R/C signal loss (possibly transient condition)         //
1520  //  ON:            Low Batt (VBATT < VBWARN)                              //
1521  ////////////////////////////////////////////////////////////////////////////
1522
1523      BYTE VB=0;
1524      BYTE previousVerbosity=0;
1525
1526      previousVerbosity = GetVerbosity();
1527      SetVerbosity(VERBOSITY_LEVEL_ERROR | VERBOSITY_LEVEL_WARNING);
1528      // Reset counter and do right shift (with wrap) on led scheduler.
1529      ledshift >>= 1; if(!ledshift) ledshift = 0x80000000;
1530
1531      // Determine appropriate LED1 pattern
1532      GetGPIO("traction voltage sensor",&VB);
1533
1534      /* LED1 ON: motor is ready to go*/
1535      if( (!inLockdown) && ( (movementType!=HILLHOLD) && (autoMode) ) && VB && (!SigmaDError)
          )
1536          grn1pattern = LED_MTR_STALL;
1537      /* Not used for now
1538      else if ( (movementType==NEUTRAL) && (!manualMode) )
1539          grn1pattern = LED_MTR_TGA;
1540      else if ( ((movementType==FORWARD) || (movementType==REVERSE)) && (!manualMode) )
1541          grn1pattern = LED_MTR_OPENLP;*/
1542      /*LED1 OFF: some condition is preventing the led to go*/
1543      else
1544          grn1pattern = LED_MTR_OFF;
1545
1546      // Determine appropriate LED2 pattern
1547      if(!Power2)                              grn2pattern = LED_MTR_OFF;
1548      else if(Mtr2_Flags1 & tga_Msk)           grn2pattern = LED_MTR_TGA;
1549      else if(V2)                              grn2pattern = LED_MTR_OPENLP;
1550      else                                     grn2pattern = LED_MTR_STALL;
1551
1552      // Determine appropriate LED3 pattern
1553      if(LedErr==0)                            redpattern = LED_FULLOFF;
1554      else if(LedErr & Lckmsk)                 redpattern = ROVIM_T2D_LED_LOCKDOWN;
1555      else if(LedErr & (SL1msk) + SL2msk)      redpattern = LED_SIGNAL_LOSS;
1556      else if(LedErr & VBATTmsk)               redpattern = LED_VBATT;
1557
1558
1559      // Adjust LED's as required.
1560      if(ledshift & grn1pattern) _LED1_ON; else _LED1_OFF;
1561      if(ledshift & grn2pattern) _LED2_ON; else _LED2_OFF;
1562      if(ledshift & redpattern)  _LED3_ON; else _LED3_OFF;
1563      SetVerbosity(previousVerbosity);
1564  }
1565
1566  //-------------------------------Commands accessible from the command line or I2C
          ----------------------
1567
1568  BYTE ROVIM_T2D_Lockdown(void)
```

```
1569 {
1570     if(UnlockingBrake)
1571     {
1572         STATUS_MSG("Aborting current emergency brake unlock.\r\n");
1573         inLockdown=FALSE;
1574         LedErr &= ~Lckmsk;
1575         UnlockingBrake=FALSE;
1576     }
1577
1578     if(inLockdown)
1579     {
1580        STATUS_MSG("Already in Lockdown mode.\r\n");
1581        return NoErr;
1582     }
1583
1584     STATUS_MSG("Going to lock down mode.\r\n");
1585
1586     //Once we engage the handbrake, the traction motor (the most critical) cannot work, so
          we're safe
1587     ROVIM_T2D_LockBrake();
1588
1589     //Stop motors controlled through dalf's firmware
1590     SoftStop(2);
1591     //Stop traction motor and set it to hold position
1592     SoftStop(3);
1593
1594     LockCriticalResourcesAccess();
1595
1596     inLockdown=TRUE;
1597     LedErr|=Lckmsk;
1598     STATUS_MSG("ROVIM now in lockdown mode. Emergency brake will finish locking and vehicle
          will \
1599 not be able to move while on this state.\r\n");
1600
1601     return NoErr;
1602 }
1603
1604 BYTE ROVIM_T2D_ReleaseFromLockdown(void)
1605 {
1606     if(!inLockdown)
1607     {
1608         STATUS_MSG("ROVIM is already good to go.\r\n");
1609         return NoErr;
1610     }
1611     if(UnlockingBrake)
1612     {
1613         STATUS_MSG("ROVIM is already ready - you have to manually unlock the emergency brake
               now.\r\n");
1614         return NoErr;
1615     }
1616
1617     if(!ROVIM_T2D_ValidateState())
1618     {
1619         ERROR_MSG("ROVIM is not ready to go. make sure \
1620 every safety system is OK and try again.\r\nFor a list of the safety checks to pass before
      the \
1621 vehicle can move, consult the user manual, or activate debug messages.\r\n");
1622         return eErr;
1623     }
1624
1625     STATUS_MSG("Manually unclamp the emergency brake within %d ms, using the identified
          button \
1626 on the control panel.\r\n\You must press it for longer than %d ms to deactivate all safety \
1627 systems.\r\nThe system will become operational once it detects it is fully unclamped.\r\n",
1628 ROVIM_T2D_BRAKE_UNLOCK_TIMEOUT, ROVIM_T2D_BRAKE_CLAMP_TIME);
1629     ROVIM_T2D_UnlockBrake();
1630     UnlockingBrake=TRUE;
1631
1632     return NoErr;
1633 }
1634
1635 BYTE ROVIM_T2D_SoftStop(void)
1636 {
```

```
1637      BYTE mtr=0;
1638
1639      if (ARGN > 2)
1640      {
1641          return eNumArgsErr;
1642      }
1643
1644      if (ARGN==2)
1645      {
1646          mtr=ARG[1];
1647          if(mtr>2)
1648          {
1649              ERROR_MSG("Motor number must be %d (traction) or %d (direction). If no motor is
                      specified, all will be stopped.\r\n",1,2);
1650              return eParmErr;
1651          }
1652          if(mtr==1) mtr=3;    //motor 1 doesn't really exist
1653          SoftStop(mtr);
1654          STATUS_MSG("Motor %d stopped.\r\n", mtr);
1655      }
1656      else
1657      {
1658          SoftStop(2);
1659          SoftStop(3);
1660          STATUS_MSG("Motors stopped.\r\n");
1661      }
1662
1663      return NoErr;
1664  }
1665
1666  /*Allows access to GPIO sub-driver*/
1667  BYTE ROVIM_T2D_ControlGPIO(void)
1668  {
1669      BYTE GPIONumber=0;
1670      BYTE value=0;
1671      BYTE option=0;
1672
1673      if (ARGN != 3)
1674      {
1675          return eNumArgsErr;
1676      }
1677      GPIONumber=ARG[2];
1678      option=ARG[1];
1679      if ((GPIONumber == 0) || (GPIONumber > ngpios))
1680      {
1681          ERROR_MSG("GPIO %d doesn't exist.\r\n",GPIONumber);
1682          return eParmErr;
1683      }
1684
1685      GPIONumber--;
1686
1687      switch(option)
1688      {
1689          case 1:
1690              SetGPIO(GPIOsDescription[GPIONumber].name);
1691              STATUS_MSG("\"%HS\" value set to 1.\r\n",GPIOsDescription[GPIONumber].name);
1692              return NoErr;
1693          case 2:
1694              ResetGPIO(GPIOsDescription[GPIONumber].name);
1695              STATUS_MSG("\"%HS\" value reset to 0.\r\n",GPIOsDescription[GPIONumber].name);
1696              return NoErr;
1697          case 3:
1698              ToggleGPIO(GPIOsDescription[GPIONumber].name);
1699              STATUS_MSG("\"%HS\" value toggled.\r\n",GPIOsDescription[GPIONumber].name);
1700              return NoErr;
1701          case 4:
1702              GetGPIO(GPIOsDescription[GPIONumber].name, &value);
1703              STATUS_MSG("\"%HS\" value is %d.\r\n",GPIOsDescription[GPIONumber].name,value);
1704              return NoErr;
1705          default:
1706              ERROR_MSG("GPIO action not valid.\r\\n");
1707              return eParmErr;
1708      }
```

```
1709 }
1710
1711 BYTE ROVIM_T2D_Accelerate(void)
1712 {
1713     BYTE dutyCycle=0;
1714
1715     if (ARGN != 2)
1716     {
1717         return eNumArgsErr;
1718     }
1719
1720     dutyCycle=ARG[1];
1721     if (dutyCycle > 100)
1722     {
1723         ERROR_MSG("Duty cycle can only vary between 0 and 100%%.\r\n");
1724         return eParmErr;
1725     }
1726
1727     if (!autoMode)
1728     {
1729         ERROR_MSG("You cannot control the vehicle with the micro controller while on manual
                mode.\r\n");
1730         return eParmErr;
1731     }
1732
1733     AccDutyCycle=dutyCycle;
1734     //Used to variate PWM led of motor1, to have a visual traction power indicator.
1735     ROVIM_T2D_LightPWMLed(AccDutyCycle, FORWARD);
1736     if (DccDutyCycle!=0)
1737     {
1738         //Wait 2*tau before starting to accelerate
1739         WaitForAccDccDecay=2*ROVIM_T2D_TIME_CONSTANT_ACC_DCC_DAC/
                ROVIM_T2D_PWM_REFRESH_PERIOD;
1740         DEBUG_MSG("Waiting for %d ms to avoid raising accelerator and decelerator
                simultaneously. \
1741 WaitForAccDccDecay=%u.\r\n", WaitForAccDccDecay, (WaitForAccDccDecay*
    ROVIM_T2D_PWM_REFRESH_PERIOD));
1742     }
1743     DccDutyCycle=0;
1744     /* If we don't reset the PWM period counter, we may have the duty cycle raise up to twice
            our spec
1745     (worst case) if the change happens when this counter is already very ahead. This way,
            worst case
1746     is the first run of the thread (ROVIM_T2D_PWM_REFRESH_PERIOD ms longer than spec).*/
1747     PeriodCnt=100;
1748     STATUS_MSG("Accelerator set to %d%%.\r\n",AccDutyCycle);
1749
1750     return NoErr;
1751 }
1752
1753 BYTE ROVIM_T2D_Decelerate(void)
1754 {
1755     BYTE dutyCycle=0;
1756
1757     if (ARGN != 2)
1758     {
1759         return eNumArgsErr;
1760     }
1761     dutyCycle=ARG[1];
1762     if (dutyCycle > 100)
1763     {
1764         ERROR_MSG("Duty cycle can only vary between 0 and 100%%.\r\n");
1765         return eParmErr;
1766     }
1767
1768     if (!autoMode)
1769     {
1770         ERROR_MSG("You cannot control the vehicle with the micro controller while on manual
                mode.\r\n");
1771         return eParmErr;
1772     }
1773
1774     DccDutyCycle=dutyCycle;
```

```
1775      ROVIM_T2D_LightPWMLed(DccDutyCycle, REVERSE);
1776      if (AccDutyCycle!=0)
1777      {
1778          //Wait 2*tau before starting to decelerate
1779          WaitForAccDccDecay=2*ROVIM_T2D_TIME_CONSTANT_ACC_DCC_DAC/
              ROVIM_T2D_PWM_REFRESH_PERIOD;
1780          DEBUG_MSG("Waiting for %d ms to avoid raising accelerator and decelerator
              simultaneously. \
1781  WaitForAccDccDecay=%u.\r\n", WaitForAccDccDecay, (WaitForAccDccDecay*
      ROVIM_T2D_PWM_REFRESH_PERIOD));
1782      }
1783      AccDutyCycle=0;
1784      /* If we don't reset the PWM period counter, we may have the duty cycle raise up to twice
            our spec
1785      (worst case) if the change happens when this counter is already very ahead. This way,
            worst case
1786      is the first run of the thread (ROVIM_T2D_PWM_REFRESH_PERIOD ms longer than spec).*/
1787      PeriodCnt=100;
1788      STATUS_MSG("Decelerator set to %d%%.\r\n",DccDutyCycle);
1789
1790      return NoErr;
1791  }
1792
1793  BYTE ROVIM_T2D_SetMovement(void)
1794  {
1795      BYTE speed=0;    //in km/h/10
1796      BYTE direction=0;
1797
1798      switch (ARGN)
1799      {
1800          case 1:
1801              //Hold (handbrake) - default: No need for any arguments
1802              direction=HILLHOLD;
1803              speed=SPEEDZERO;
1804              break;
1805          case 2:
1806              //Neutral or hold: specify only the type of movement - speed is zero
1807              direction=ARG[1];
1808              speed=SPEEDZERO;
1809              break;
1810          case 3:
1811              //Any kind of movement: specify the type of movement and the desired speed
1812              direction=ARG[1];
1813              speed=ARG[2];
1814              break;
1815          default:
1816              DEBUG_MSG("Wrong number of arguments.\r\n");
1817              return eNumArgsErr;
1818      }
1819
1820      if (direction > 3)
1821      {
1822          ERROR_MSG("Direction must be %d (forward), %d (reverse), %d (neutral) or %d (hold).\
                r\n",
1823          FORWARD, REVERSE, NEUTRAL, HILLHOLD);
1824          return eParmErr;
1825      }
1826      if ( (speed > ROVIM_T2D_MAX_SPEED) || ((direction <2) && (speed <
            ROVIM_T2D_LOWER_SPEED_LIMIT)) )
1827      {
1828          //in neutral and hold cases, speed is allways 0, wich may be <
                ROVIM_T2D_LOWER_SPEED_LIMIT
1829          ERROR_MSG("Speed must vary in 1/10 km/h between %d km/h/10 and %d km/h/10.\r\n",
1830          ROVIM_T2D_LOWER_SPEED_LIMIT, ROVIM_T2D_MAX_SPEED);
1831          DEBUG_MSG("speed=%d, direction=%d.\r\n",speed, direction);
1832          return eParmErr;
1833      }
1834      if ( (direction!=HILLHOLD) && (!autoMode) )
1835      {
1836          ERROR_MSG("You cannot move the vehicle with the micro controller while on manual
                mode.\r\n");
1837          return eParmErr;
1838      }
```

```
1839     if ( (direction!=HILLHOLD) && (SigmaDError) )
1840     {
1841         ERROR_MSG("There is an error with the traction controller, or it is turned OFF. Wait
                %ld \
1842 ms after solving the error before retrying.\r\n",(long)ROVIM_T2D_SIGMAD_ERROR_TIMEOUT);
1843         return eParmErr;
1844     }
1845
1846     switch(direction)
1847     {
1848         case HILLHOLD: //Hold
1849             SetGPIO("engage forward");
1850             SetGPIO("engage reverse");
1851             SetGPIO("engage handbrake");
1852             ResetGPIO("activate traction");
1853
1854             movementType=HILLHOLD;
1855             desiredMovement.type=HILLHOLD;
1856             desiredMovement.speed=SPEEDZERO;
1857             CMD='G';
1858             ARG[0]=ROVIM_T2D_ACCELERATE_CMD_CODE;
1859             ARG[1]=0;
1860             ARGN=2;
1861             TeCmdDispatchExt();
1862             /*The trace for this case is debug, but for the others is status, because this
                    command is
1863             called from anywhere is the code to make sure the outputs are as expected. This
                    creates
1864             some unexpected messages during normal program execution. The others cases do
                    not have this problem*/
1865             DEBUG_MSG("Setting vehicle on hill hold. It will hold position once it reaches a
                     standstill.\r\n");
1866             DEBUG_MSG("desired type=%d, speed=%d, current type=%d.\r\n",desiredMovement.type
                    , desiredMovement.speed, movementType);
1867             break;
1868         case NEUTRAL: //Neutral
1869             ResetGPIO("engage handbrake");
1870             SetGPIO("engage forward");
1871             SetGPIO("engage reverse");
1872             SetGPIO("activate traction");
1873
1874             movementType=NEUTRAL;
1875             desiredMovement.type=NEUTRAL;
1876             desiredMovement.speed=SPEEDZERO;     //actually, we don't care
1877             CMD='G';
1878             ARG[0]=ROVIM_T2D_ACCELERATE_CMD_CODE;
1879             ARG[1]=0;
1880             ARGN=2;
1881             TeCmdDispatchExt();
1882             STATUS_MSG("Vehicle set in neutral.\r\n");
1883             DEBUG_MSG("desired type=%d, speed=%d, current type=%d.\r\n",desiredMovement.type
                    , desiredMovement.speed, movementType);
1884             break;
1885         case FORWARD: //Forward
1886             WARNING_MSG("Speed control done only in open loop currently. Check manually if
                    the set speed matches your request.\r\n");
1887             desiredMovement.type=FORWARD;
1888             desiredMovement.speed=speed;
1889             if ((vel1) && (movementType!=desiredMovement.type))
1890             {
1891                 DEBUG_MSG("Bringing vehicle to a standstill before moving forward. Please
                        wait.\r\n");
1892                 ROVIM_T2D_FullBrake();
1893             }
1894             if (movementType==desiredMovement.type)
1895             {
1896                 //If we're already moving in the direction we want, just accelerate
1897                 ROVIM_T2D_SetSpeed(desiredMovement.speed);
1898                 STATUS_MSG("Vehicle moving forward at approximately %d km/h/10.\r\n",
                        desiredMovement.speed);
1899             }
1900             DEBUG_MSG("desired type=%d, speed=%d, current type=%d.\r\n",desiredMovement.type
                    , desiredMovement.speed, movementType);
```

```
1901                break;
1902            case REVERSE: //Reverse
1903                WARNING_MSG("Speed control done only in open loop currently. Check manually if
                        the set speed matches your request.\r\n");
1904                desiredMovement.type=REVERSE;
1905                desiredMovement.speed=speed;
1906                if ((vel1) && (movementType!=desiredMovement.type))
1907                {
1908                    DEBUG_MSG("Bringing vehicle to a standstill before moving in reverse. Please
                            wait.\r\n");
1909                    ROVIM_T2D_FullBrake();
1910                }
1911                if (movementType==desiredMovement.type)
1912                {
1913                    //If we're already moving in the direction we want, just accelerate
1914                    ROVIM_T2D_SetSpeed(desiredMovement.speed);
1915                    STATUS_MSG("Vehicle moving backwards at approximately %d km/h/10.\r\n",
                            desiredMovement.speed);
1916                }
1917                DEBUG_MSG("desired type=%d, speed=%d, current type=%d.\r\n",desiredMovement.type
                        , desiredMovement.speed, movementType);
1918                break;
1919            default:
1920                ERROR_MSG("Unexpected argument value.\r\n");
1921                break;
1922        }
1923
1924        return NoErr;
1925    }
1926
1927    BYTE ROVIM_T2D_Turn(void)
1928    {
1929        BYTE fullAngle=0;
1930        long centerAngle=0;
1931        short long y=0;
1932        long temp=0;
1933
1934        if ((ARGN != 2) && (ARGN != 1))
1935        {
1936            return eNumArgsErr;
1937        }
1938        if (ARGN == 1)
1939        {
1940            SoftStop(2);
1941            STATUS_MSG("Direction motor stopped.\r\n");
1942            return NoErr;
1943        }
1944
1945        fullAngle=ARG[1];
1946        if (fullAngle > ROVIM_T2D_DIR_ANGULAR_RANGE)
1947        {
1948            ERROR_MSG("Turn angle can only vary between 0º (full port) and %dº (full starboard)
                    .\r\n", ROVIM_T2D_DIR_ANGULAR_RANGE);
1949            return eParmErr;
1950        }
1951        if (!vel1)
1952        {
1953            ERROR_MSG("Vehicle must be moving when turning. Set the vehicle to move and retry.\r
                    \n");
1954            return eErr;
1955        }
1956
1957        if (!autoMode)
1958        {
1959            ERROR_MSG("You cannot move the vehicle with the micro controller while on manual
                    mode.\r\n");
1960            return eParmErr;
1961        }
1962        /*Since the direction range is not evenly balanced (it turns more to one side than to
                the other
1963        we use the center angle for intermediate calculations, to get a more accurate result.
1964        Also, to the user, it's the angle relative to straight line movement the most important
                */
```

```
1965      centerAngle=(long)ROVIM_T2D_DIR_ANGULAR_RANGE>>1;
1966      centerAngle=(long)fullAngle-centerAngle;
1967      if (centerAngle > 0)
1968      {
1969          STATUS_MSG("Turning vehicle %ld° to port (%ld° full angle).\r\n",centerAngle, (long)
                fullAngle);
1970      }
1971      else if (centerAngle == 0)
1972      {
1973          STATUS_MSG("Pointing vehicle in a straight line (%ld° full angle).\r\n", (long)
                fullAngle);
1974      }
1975      else
1976      {
1977          STATUS_MSG("Turning vehicle %ld° to starboard (%ld° full angle).\r\n", (long)abs(
                centerAngle),(long)fullAngle);
1978      }
1979
1980      temp=(long) (ROVIM_T2D_DIR_TICK_UPPER_LIMIT-ROVIM_T2D_DIR_TICK_LOWER_LIMIT)*centerAngle
            /**2*/;
1981      temp=(long) temp/ROVIM_T2D_DIR_ANGULAR_RANGE/*/2*/;
1982      temp=(long) temp + ROVIM_T2D_DIR_CENTER_TICK_CNT;
1983
1984      if(temp<ROVIM_T2D_DIR_TICK_LOWER_LIMIT)
1985      {
1986         DEBUG_MSG("target position out of lower bound. This may happen if the limits are
                unbalanced, \
1987 or due to erroneous calculations. full turn angle=%ld, bound=%d.\r\n",temp,
      ROVIM_T2D_DIR_TICK_LOWER_LIMIT);
1988          temp=ROVIM_T2D_DIR_TICK_LOWER_LIMIT;
1989      }
1990      if(temp>ROVIM_T2D_DIR_TICK_UPPER_LIMIT)
1991      {
1992          DEBUG_MSG("target position out of upper bound. This may happen if the limits are
                unbalanced, \
1993 or due to erroneous calculations. full turn angle=%ld, bound=%d.\r\n",temp,
      ROVIM_T2D_DIR_TICK_UPPER_LIMIT);
1994          temp=ROVIM_T2D_DIR_TICK_UPPER_LIMIT;
1995      }
1996      y=(short long) temp;
1997
1998      DEBUG_MSG("y=%ld, temp=%ld, ang range=%d, tick range=%d, angle=%d, VMID2=%d, ACC2=%d.\r\
            n",
1999      (long)y, (long)temp, (BYTE) ROVIM_T2D_DIR_ANGULAR_RANGE,
2000      (BYTE)(ROVIM_T2D_DIR_TICK_UPPER_LIMIT-ROVIM_T2D_DIR_TICK_LOWER_LIMIT), (BYTE)
            centerAngle, VMID2, ACC2);
2001
2002      MoveMtrClosedLoop(2, y, VMID2, ACC2);
2003
2004      return NoErr;
2005  }
2006
2007  BYTE ROVIM_T2D_DebugControl(void)
2008  {
2009      BYTE option=0;
2010      BYTE resetNotKick=0;
2011      BYTE accNotDcc=0;
2012      BYTE dutyCycle=0;
2013      BYTE verbosity=0;
2014      BYTE temp=0;
2015      BYTE scalingFactor=0;
2016
2017      if (ARGN < 2)
2018      {
2019          return eNumArgsErr;
2020      }
2021      option=ARG[1];
2022      switch(option)
2023      {
2024          case 1: //Get verbosity level
2025              verbosity=GetVerbosity();
2026              FATAL_ERROR_MSG("Not really an error.\r\nCurrent verbosity=%d.\r\n", verbosity);
2027              break;
```

```
2028            case 2: //Set verbosity level
2029                if(ARGN != 3)
2030                {
2031                    ERROR_MSG("You must specify the verbosity you want to set.\r\n");
2032                    return eNumArgsErr;
2033                }
2034                verbosity=ARG[2];
2035                SetVerbosity(verbosity);
2036                FATAL_ERROR_MSG("Not really an error.\r\nCurrent verbosity=%d.\r\n", verbosity);
2037                break;
2038            case 3: //Toggle between automatic (default) and manual execution of the System
                        monitoring task
2039                ManualSysMonitoring=~ManualSysMonitoring;
2040                if(ManualSysMonitoring)
2041                {
2042                    STATUS_MSG("ROVIM T2D system monitoring done manually now. Use this command
                                \
2043    with the respective argument to run another time the T2D system monitoring task.\r\n");
2044                }
2045                else
2046                {
2047                    STATUS_MSG("ROVIM T2D system monitoring reverted to default (automatic).\r\n
                            ");
2048                }
2049                break;
2050            case 4: //run the system monitor task, if in manual mode
2051                if(ManualSysMonitoring)
2052                {
2053                    STATUS_MSG("Running ROVIM T2D system monitoring one time.\r\n");
2054                    ROVIM_T2D_MonitorSystem();
2055                }
2056                else
2057                {
2058                    ERROR_MSG("ROVIM T2D system monitoring is not set to manual mode.\r\n");
2059                }
2060                break;
2061            case 5:
2062                if (ARGN == 5)
2063                {
2064                    DebugPWM=ARG[2]+1;
2065                    accNotDcc=ARG[3];
2066                    dutyCycle=ARG[4];
2067                    STATUS_MSG("Running ROVIM T2D PWM generator debug with period count %d. ACC
                                ?:%d, DCC?:%d, \
2068    Dutycyle=%d.\r\n",(DebugPWM-1), accNotDcc, (!accNotDcc), dutyCycle);
2069                    CMD='G';
2070                    ARG[0]=accNotDcc?ROVIM_T2D_ACCELERATE_CMD_CODE:ROVIM_T2D_DECELERATE_CMD_CODE
                                ;
2071                    ARG[1]=dutyCycle;
2072                    ARGN=2;
2073                    TeCmdDispatchExt();
2074                    break;
2075                }
2076                else if (ARGN == 3)
2077                {
2078                    DebugPWM=ARG[2]+1;
2079                    STATUS_MSG("Running ROVIM T2D PWM generator debug with period count %d.\r\n"
                                ,(DebugPWM-1));
2080                    break;
2081                }
2082                ERROR_MSG("You have to specify the number of periods you want to see the PWM
                            generator \
2083    debug info and (optional), weather you want to accelerate or decelerate and the desired duty
            cycle.\
2084     The PWM generator is not controlled by this command, only it's debug information. You may
            have to\
2085     activate debug traces.\r\n");
2086                return eNumArgsErr;
2087            case 6: //Control the watchdog
2088                #ifdef WATCHDOG_ENABLED
2089                if (ARGN < 3)
2090                {
```

```
2091                ERROR_MSG("You must specify if you want to kick the watchdog or reset the
                        system through it.\r\n");
2092                return eNumArgsErr;
2093            }
2094            resetNotKick=ARG[2];
2095            if(resetNotKick)
2096            {
2097                STATUS_MSG("Resetting system through watchdog.\r\n");
2098                HardReset();
2099            }
2100            else
2101            {
2102                STATUS_MSG("Kicking watchdog. Kick period=%d ms. For actual watchdog timer
                        timeout period, consult config.h\r\n",WATCHDOG_PERIOD);
2103                KickWatchdog();
2104            }
2105            break;
2106            #else //WATCHDOG_ENABLED
2107            STATUS_MSG("Watchdog is disabled. Recompile.\r\n");
2108            break;
2109            #endif //WATCHDOG_ENABLED
2110        case 7: //Lock/Unlock resources
2111            ResourcesLockFlag=~ResourcesLockFlag;
2112            STATUS_MSG("Resources Lock Flag toggled to=%d.\r\n",ResourcesLockFlag);
2113            break;
2114        case 8: //Calibrate speed to duty cycle conversion
2115            if (ARGN == 3)
2116            {
2117                scalingFactor=ARG[2];
2118                SpeedDCScaling=scalingFactor;
2119                STATUS_MSG("Speed to duty cycle scaling factor set to %d.\r\n",(
                        SpeedDCScaling/10) );
2120                break;
2121            }
2122            else
2123            {
2124                STATUS_MSG("Current speed to duty cycle scaling factor: %d.\r\nCurrent LUT:\
                        r\n",(SpeedDCScaling/10) );
2125                /*for(i=0;i<SpeedToDutyCycleLen;i++)
2126                {
2127                    MSG("%d, ",SpeedToDutyCycle[i]);
2128                }
2129                MSG("\r\nLUT size=%d.\r\n",SpeedToDutyCycleLen);*/
2130                break;
2131            }
2132        case 9: //Manipulate error information print control variables
2133            ForcePrintMsg=TRUE;
2134            STATUS_MSG("Printing errors on periodic tasks one time.\r\n");
2135            break;
2136        default:
2137            ERROR_MSG("Option does not exist.\r\n");
2138            break;
2139    }
2140    return NoErr;
2141 }
```

```
1  /*************************************************************************
2  **************************************************************************
3  **
4  **
5  **                  rovim_t2d.h - Definitions for the "tracção, travagem e
6  **                          direcção" (T2D) module of the ROVIM project.
7  **
8  **      This module builds on and extends the firmware of the Dalf-1F motor
9  **      control board to implement the T2D module of the ROVIM project.
10 **
11 **      It comprises, for the T2D, its describing structures and
12 **      definitions.
13 **
14 **      This code was designed originally for the Dalf-1F motor control
15 **      board, the brain of the T2D module.
16 **      Original Dalf-1F firmware revision was 1.73.
17 **      See Dalf-1F owner's manual and the ROVIM T2D documentation for more
```

```
18 **       details.
19 **
20 **           The ROVIM Project
21 ********************************************************************************
22 ********************************************************************************/
23
24 #ifndef __ROVIM_T2D_H
25 #define __ROVIM_T2D_H
26 #include     "dalf.h"
27
28 //Function prototypes
29 void ROVIM_T2D_Init(void);
30 void ROVIM_T2D_Start(void);
31 void ROVIM_T2D_ConfigGPIOs(void);
32 void ROVIM_T2D_Greeting(void);
33 void ROVIM_T2D_MonitorSystem(void);
34 BOOL ROVIM_T2D_LockBrake(void);
35 BOOL ROVIM_T2D_UnlockBrake(void);
36 BOOL ROVIM_T2D_ValidateState(void);
37 void ROVIM_T2D_ServicePWM(void);
38 BOOL ROVIM_T2D_FinishReleaseFromLockdown(void);
39 void ROVIM_T2D_MonitorSystem(void);
40 BOOL ROVIM_T2D_DetectTractionEror(BYTE period);
41 BOOL ROVIM_T2D_DetectBrakingError(BYTE period);
42 BOOL ROVIM_T2D_DetectDirectionEror(BYTE period);
43 void ROVIM_T2D_MonitorTractionWarning(BYTE period);
44 void ROVIM_T2D_MonitorBrakingWarning(BYTE period);
45 void ROVIM_T2D_MonitorDirectionWarning(BYTE period);
46 BOOL ROVIM_T2D_DetectFatalError(WORD period);
47 BOOL ROVIM_T2D_DetectBrakeUnlock(WORD period);
48 void ROVIM_T2D_LockUnusedResourcesAccess(void);
49 void ROVIM_T2D_ConfigSerialPort(void);
50 void ROVIM_T2D_UpdateVelocity1(void);
51 void ROVIM_T2D_ConfigDefaultParamBlock(void);
52 void ROVIM_T2D_ConfigDirParamBlock(void);
53 void ROVIM_T2D_LockCriticalResourcesAccess(void);
54 void ROVIM_T2D_UnlockCriticalResourcesAccess(void);
55 BOOL ROVIM_T2D_IsCommandLocked(BYTE cmd);
56 void ROVIM_T2D_MonitorManualMode(BYTE period);
57 void ROVIM_T2D_PendingCmd(BYTE period);
58 void ROVIM_T2D_FullBrake(void);
59 void ROVIM_T2D_SetSpeed(BYTE speed);
60 BYTE ROVIM_T2D_SoftStop(void);
61 BYTE ROVIM_T2D_LightPWMLed(BYTE dutyCycle, BYTE direction);
62 void ROVIM_T2D_DetectSigmaDError(BYTE period);
63 void     ROVIM_T2D_ServiceLED(void);
64
65 //functions accessible from the command line
66 BYTE ROVIM_T2D_CmdDispatch(void);
67 BYTE ROVIM_T2D_Lockdown(void);
68 BYTE ROVIM_T2D_ReleaseFromLockdown(void);
69 BYTE ROVIM_T2D_ControlGPIO(void);
70 BYTE ROVIM_T2D_Accelerate(void);
71 BYTE ROVIM_T2D_Decelerate(void);
72 BYTE ROVIM_T2D_SetMovement(void);
73 BYTE ROVIM_T2D_Turn(void);
74 BYTE ROVIM_T2D_DebugControl(void);
75
76 extern WORD     ROVIM_T2D_sysmonitorcount;              // ROVIM T2D system state monitoring
       timeout counter;
77 extern WORD     ROVIM_T2D_pwmrefreshcount;              // ROVIM T2D PWM refresh timeout
       counter;
78 extern BOOL     ManualSysMonitoring;
79
80 extern BOOL inLockdown;
81 extern BOOL autoMode;
82 extern BOOL SigmaDError;
83 extern BYTE movementType;
84
85 extern long vel2;
86 extern long acc1;
87 extern long vel1;
88
```

```
 89 typedef struct{
 90     BYTE type;
 91     BYTE speed;
 92 }movement;
 93
 94 //definitions
 95 //command codes
 96 #define ROVIM_T2D_LOCKDOWN_CMD_CODE           (CUSTOM_CMD_ID_OFFSET)
 97 #define ROVIM_T2D_RELEASE_CMD_CODE            (CUSTOM_CMD_ID_OFFSET+1)
 98 #define ROVIM_T2D_SOFTSTOP_CMD_CODE           (CUSTOM_CMD_ID_OFFSET+2)
 99 #define ROVIM_T2D_CONTROL_GPIO_CMD_CODE       (CUSTOM_CMD_ID_OFFSET+3)
100 #define ROVIM_T2D_ACCELERATE_CMD_CODE         (CUSTOM_CMD_ID_OFFSET+4)
101 #define ROVIM_T2D_DECELERATE_CMD_CODE         (CUSTOM_CMD_ID_OFFSET+5)
102 #define ROVIM_T2D_SET_MOVEMENT_CMD_CODE       (CUSTOM_CMD_ID_OFFSET+6)
103 #define ROVIM_T2D_TURN_CMD_CODE               (CUSTOM_CMD_ID_OFFSET+7)
104 #define ROVIM_T2D_DEBUG_CTRL_CMD_CODE         (CUSTOM_CMD_ID_OFFSET+8)
105
106 #define SigmaDLed ADC0[4]
107 #define DirPos    ADC0[3]
108 /*time it takes for Sigma Drive external error led to be off to assume the error is cleared.
109 When there is an error, the controller blinks the error code, then pauses for about 2s
        before repeating*/
110 #define ROVIM_T2D_SIGMAD_ERROR_TIMEOUT 2000
111 //digital filter cut-off frequency parameter
112 #define ROVIM_T2D_DECAY 0x80     //fc=~25Hz
113 #define ROVIM_T2D_FENBL 0x54
114 //voltage threshold to produce a warning in dalf firmware
115 #define ROVIM_T2D_VBWARN 12000
116 //Traction system related definitions
117 //Minimum duty cycle and duty cycle increase. This should be a divider of 100(%)
118 //#define ROVIM_T2D_PWM_MIN 2
119 //relation between PWM period and duty cycle
120 //#define ROVIM_T2D_PWM_CNTS_PER_PERIOD 100
121 //time constant of the low pass RC filter converting the accelerator and decelerator signals
        to analogue, in ms (47k*47u)
122 #define ROVIM_T2D_TIME_CONSTANT_ACC_DCC_DAC 220
123 //traction PWM signal refresh period, in ms
124 #define ROVIM_T2D_PWM_REFRESH_PERIOD 1
125 //maximum speed the user can order the vehicle to move, in Km/h/10
126 #define ROVIM_T2D_MAX_SPEED 45
127 //maximum speed the vehicle can achieve at any point in time, in Km/h/10
128 #define ROVIM_T2D_CRITICAL_SPEED ( ((long) ROVIM_T2D_MAX_SPEED)*12/10 )
129 //minimum speed of the vehicle, in Km/h/10
130 /*minimum speed should be !=0. The encoder is just not sensible enough bellow these speeds*/
131 #define ROVIM_T2D_LOWER_SPEED_LIMIT 5
132 //number of ticks per rev (not the #teeth of the gear where
133 //the encoder is mounted, but of the gear that revs at the same speed as the wheels)
134 #define ROVIM_T2D_TRACTION_TPR 39
135 //Average perimeter of the real wheel (depends on tire tread, load and tire pressure), in cm
        */
136 #define ROVIM_T2D_WHEEL_PERIMETER 176
137 //sincronize this period with VSP for traction encoder
138 #define ROVIM_T2D_VSP1 250   //ms
139 #define ROVIM_T2D_SYSTEM_MONITOR_PERIOD    1000//ms
140
141 #define ROVIM_T2D_VEL1_CALC_MIN_TICK_CNT 5
142
143
144 #define WATCHDOG_PERIOD 0x200          //512 ms
145 #define ROVIM_T2D_NBR 5                //57600 baud/s
146
147
148 //threshold of stressful motor operation (indicating some sort of error), in % of maximum
        PWM
149 //totally arbitrary number defined by me at this point (ah, but which point is it?? You'll
        never know. God, I need to see a shrink...)
150 #define ROVIM_T2D_DIRECTION_MOTOR_STRESS_DUTY_CYCLE_THRESHOLD 80      //% of VMAX
151
152 //threshold of stressful situations measured on the motor to formally declare an error
153 #define ROVIM_T2D_DIRECTION_MOTOR_STRESS_CNT_THRESHOLD 1024 //ex: at 0,2s sample time, it
        gives ~3 m of continuous operation at high stress before declaring an error
154
155 //time it takes for the brake to go from fully unclamped to fully clamped, in ms
```

```
156 #define ROVIM_T2D_BRAKE_CLAMP_TIME 6000
157 //maximum time it may take for the velocity to reach the defined error band of the final
        value on closed loop traction speed control, in ROVIM_T2D_PWM_REFRESH_PERIOD units
158 #define ROVIM_T2D_MAX_SETTLING_TIME 10000
159 //traction acceleration threshold that defines an error situation, such as crash, or short
        circuit, in Km/10/h/s
160 #define ROVIM_T2D_CRASH_ACC_THRESHOLD 50
161 //maximum time needed to stop the vehicle once it has been set on hold, in ms
162 #define ROVIM_T2D_MOVING_ON_HOLD_TIMEOUT 5000
163 /*maximum continuous time the user can be pressing the manual brake unlock button to
        successfully release from lockdown,
164 in ms*/
165 #define ROVIM_T2D_BRAKE_UNLOCK_TIMEOUT 30000
166 //time threshold of continuous press of unclamp button to declare an error, in ms.
167 #define ROVIM_T2D_FATAL_UNCLAMP_TIMEOUT  120000
168
169 //total direction safe travel, in degrees (should be an even number)
170 #define ROVIM_T2D_DIR_ANGULAR_RANGE 86  //total direction travel~90º
171 //upper tick count (potentiometer value) that the direction can safely reach
172 #define ROVIM_T2D_DIR_TICK_UPPER_LIMIT 0xB8 //touches the end-of-travel at 0xC8
173 //lower tick count (potentiometer value) that the direction can safely reach
174 #define ROVIM_T2D_DIR_TICK_LOWER_LIMIT 0x23 //touches the end-of-travel at 0x13
175 //tick count corresponding to direction mid-course (straight line movement). Used because
        tick slack isn't equal for each side of rotation
176 #define ROVIM_T2D_DIR_CENTER_TICK_CNT 0x6D
177 //direction motor mode1 flags. See dalf owners manual
178 #define ROVIM_T2D_DIR_MODE1 0x32
179 //direction motor mode2 flags. See dalf owners manual
180 #define ROVIM_T2D_DIR_MODE2 0x00
181 //direction motor mode3 flags. See dalf owners manual
182 #define ROVIM_T2D_DIR_MODE3 0x21
183 //velocity sampling period for direction motor
184 #define ROVIM_T2D_DIR_VSP 20
185 //minimum PWM duty cycle for direction control
186 #define ROVIM_T2D_DIR_MIN_PWM 30
187 //maximum PWM duty cycle for direction control
188 #define ROVIM_T2D_DIR_MAX_PWM 100
189 //maximum analog error -  not important in this application
190 #define ROVIM_T2D_DMAX 255
191 #endif /*__ROVIM_T2D_H */
```

```
1  /*****************************************************************************
2  ******************************************************************************
3  **
4  **
5  **                                    rovim.h - Description of the ROVIM sytem.
6  **
7  **              This module describes the whole ROVIM system, from the Dalf-1F
8  **              motor control board firmware point of view.
9  **
10 **              It comprises, for the system and each of its subsystems, its
11 **              describing structures and definitions.
12 **              Different non-runtime system configurations are also referenced and
13 **              chosen here.
14 **
15 **              This code was designed originally for the Dalf-1F motor control
16 **              board, the brain of the T2D module.
17 **              Original Dalf-1F firmware revision was 1.73.
18 **              See Dalf-1F owner's manual and the ROVIM T2D documentation for more
19 **              details.
20 **
21 **                        The ROVIM Project
22 ******************************************************************************
23 *****************************************************************************/
24
25 #ifndef __ROVIM_H
26 #define __ROVIM_H
27
28 //System description and default configuration
29 //#include "rovim_description.h"                                  // Description of the
       project
30 #include        "rovim_t2d.h"                                          // Description of
       the T2D subsystem
```

```
31
32 #define INIT_VERBOSITY_LEVEL 0x07    //disable call info verbosity for now, due to the issue
       with the #line directive
33
34 //Project ID
35 #define ROVIM_T2D_SW_MAJOR_ID 1
36 #define ROVIM_T2D_SW_MINOR_ID 0
37 #define ROVIM_T2D_RELEASE_DATE "18-12-2015"
38 #define ROVIM_T2D_CONTACTS "Goncalo Andre (programmer): goncalofr87@gmail.\
39 com\r\nAntonio Serralheiro (superviser): ajserralheiro@gmail.com"
40
41 #endif /*__ROVIM_H*/
```

```
 1 /*********************************************************************************
 2 *********************************************************************************
 3 **
 4 **
 5 **                  rovim_config_v0.1.h - Configuration of the ROVIM
 6 **                          system for the version 0.1 of the ROVIM
 7 **                          T2D software.
 8 **
 9 **       This file holds the non-runtime software configuration profile of
10 **       the ROVIM system for the defined software version. It may be used
11 **       for other versions too, as long as it is unchanged.
12 **
13 **       This code was designed originally for the Dalf-1F motor control
14 **       board, the brain of the T2D module.
15 **       Original Dalf-1F firmware revision was 1.73.
16 **       See Dalf-1F owner's manual and the ROVIM T2D documentation for more
17 **       details.
18 **
19 **           The ROVIM Project
20 *********************************************************************************
21 *********************************************************************************/
22
23 #ifndef __ROVIM_CONFIG_V0_1_H
24 #define __ROVIM_CONFIG_V0_1_H
25
26 //use maximum verbosity
27 #define INIT_VERBOSITY_LEVEL 0x0F    //disable call info verbosity for now, due to the issue
       with the #line directive
28
29 #endif /*__ROVIM_CONFIG_V0_1_H*/
```

B

Apêndice B - Esquemas elétricos e *layout* das placas eletrónicas

C

Apêndice C - Lista de componentes

Id.[1]	Qt.[2]	Componente	Descrição	Subsistema[3]
C1	1	*Chassis*	*Chassis*, rodas, eixo traseiro com carreto de transmissão e sistemas de travagem e viragem de moto-quatro	*Chassis*
C2	Indef.[4]	Ferro sortido	Ferro usado nas estruturas de fixação de componentes[5] e outras adaptações soldadas ao *chassis*	*Chassis*
C3	Indef.	Material de fixação	Porcas, parafusos, anilhas, anilhas de mola, cavilhas e outros materiais não discriminados de diversas medidas, usados na fixação rígida dos componentes, entre si, ou ao *chassis*	
C4	1	Plataforma inferior	Plataforma de madeira *Medium-Density Fibreboard* (MDF) cortada, furada e escareada, à medida para a estrutura inferior	*Chassis*
C5	1	Mini-plataforma inferior	Plataforma de madeira MDF cortada, furada e escareada, entre a estrutura inferior e a coluna da direção	*Chassis*
C6	2	Tábuas de madeira	Tábuas de madeira, cortadas à medida, fixadas verticalmente à parte frontal da estrutura de suporte das plataformas, para contenção das baterias	*Chassis*
C7	1	Plataforma superior bombordo	Plataforma de madeira MDF cortada, furada e escareada, à medida para o lado de bombordo da estrutura superior	*Chassis*
C8	1	Plataforma superior estibordo	Plataforma de madeira MDF cortada, furada e escareada, à medida para o lado de estibordo da estrutura superior	*Chassis*
C9	7	Cantoneiras de madeira	Cantoneiras de madeira, de tamanhos diversos, coladas à plataforma inferior, para contenção das baterias ao nível da base	*chassis*
C10	6	Baterias NP55-12R	Baterias recarregáveis seladas de ácido-chumbo, de 12 V	Baterias
C11	4	Elásticos de retenção de cargas	Elásticos, de tamanhos diversos, com ganchos de ferro nas pontas, para abraçar e conter as baterias ao nível do topo	
C12	1	Carregador de baterias 12 V	Carregador de baterias de ácido-chumbo de 12 V	
C13	1	Carregador de baterias 72 V	Carregador de baterias de ácido-chumbo de 72 V	
C14	1	Agni B-95R	Motor DC com escovas	
C15	1	Carreto de 11 dentes	Carreto compatível com o carreto instalado no veio traseiro	Redutor da tração
C16	2	Engrenagens 42 dentes	Engrenagens cilíndricas, de módulo 2, ângulo de pressão de 20°, material `C 43 UNI 7847`, de acordo com o catálogo eurocorreias 2012 [17]	Redutor da tração
C17	1	Engrenagem 21 dentes	Engrenagens cilíndricas, de módulo 2, ângulo de pressão de 20°, material `C 43 UNI 7847`, de acordo com o catálogo eurocorreias 2012 [17]	Redutor da tração
C18	1	Engrenagem 14 dentes	Engrenagens cilíndricas, de módulo 2, ângulo de pressão de 20°, material `C 43 UNI 7847`, de acordo com o catálogo eurocorreias 2012 [17]	Redutor da tração

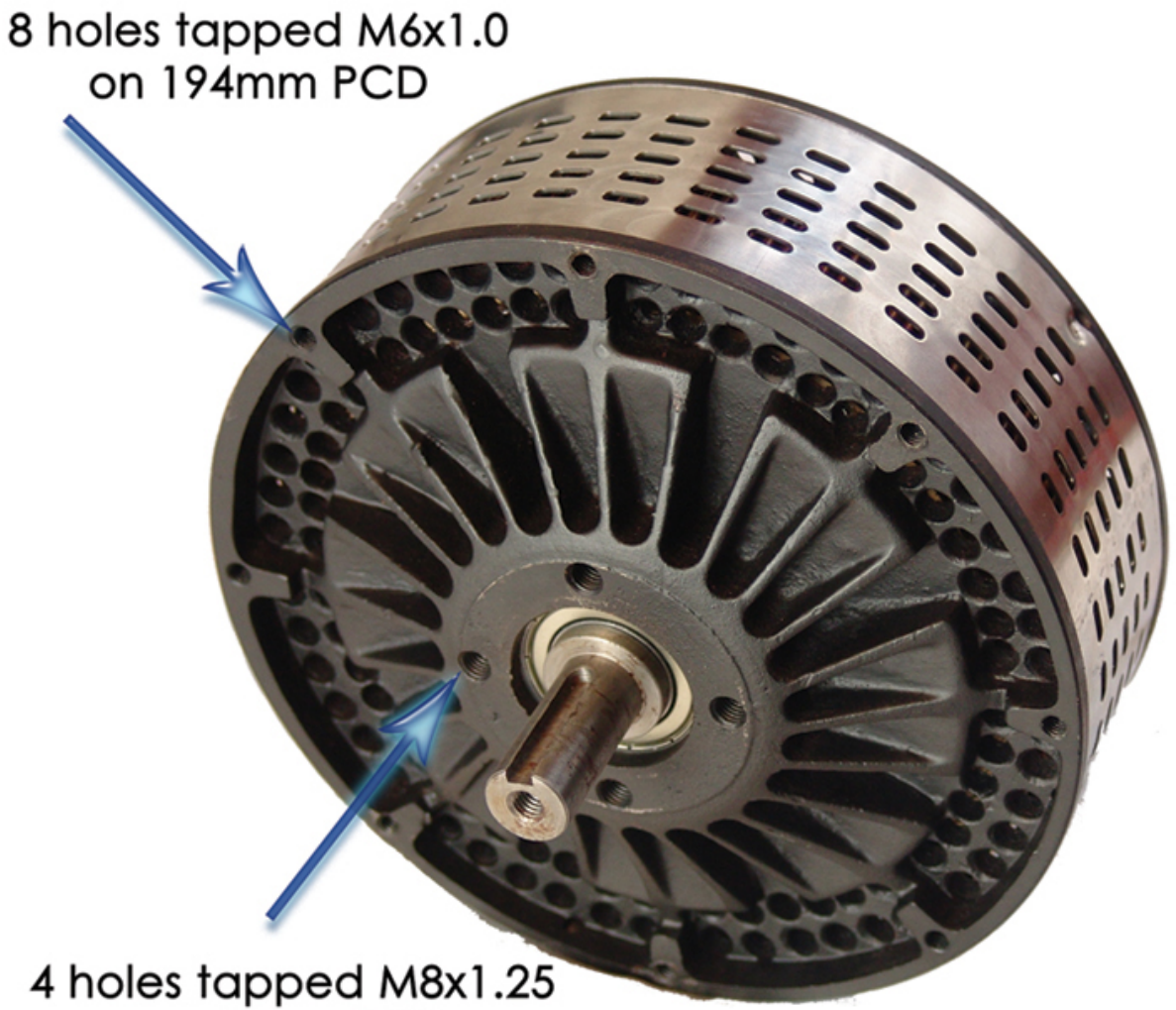
[1] Identificação
[2] Quantidade
[3] Peça a que o componente pertence, se aplicável
[4] Quantidade indefinida
[5] Que são: 1) Estrutura central de fixação das plataformas; 2) estrutura de suporte do redutor de direção; 3) estrutura de fixação do motor de tração; 4) estrutura de fixação do sensor de velocidade

C19	2	Rolamentos Koyo 6001	Rolamentos ranhurados de esferas, como especificação do catalogo Koyo [18]	Redutor da tração
C20	2	Rolamentos Koyo 6003	Rolamentos ranhurados de esferas, como especificação do catalogo Koyo [18]	Redutor da tração
C21	1	Chave de veio	Chave para fixação de engrenagem ao veio do motor de tração, de acordo com a norma ISO/R773 [16]	Redutor da tração
C22	1	Espaçador para veio	Espaçador para segurar engrenagem no veio do motor	Redutor da tração
C23	1	Fixador para carreto	Bolacha com furo para fixar carreto acoplado ao redutor no veio	Redutor da tração
C24	1	Chapa motor	Chapa estrutural de fixação dos componentes do redutor do motor, de acordo com o desenho "chapa motor"do apêndice D	Redutor da tração
C25	1	Chapa corrente	Chapa estrutural de fixação dos componentes do redutor do motor, de acordo com o desenho "chapa corrente"do apêndice D	Redutor da tração
C26	1	Veio 16 mm	Veio de fixação de engrenagens do redutor do motor, de acordo com o desenho "veio 16.12"do apêndice D	Redutor da tração
C27	1	Veio 17 mm	Veio de fixação de engrenagens do redutor do motor, de acordo com o desenho "20.23"do apêndice D	Redutor da tração
C28	Indef.	Material de fixação do redutor	Freio, parafusos, anilhas e anilhas de mola de diversas medidas usadas na fixação dos componentes do redutor do motor de tração	Redutor da tração
C29	1	Corrente	Corrente original da moto-quatro	
C30	1	Sigmadrive PMT835M	Controlador de motor DC de ímanes permanentes	**To do** (15)

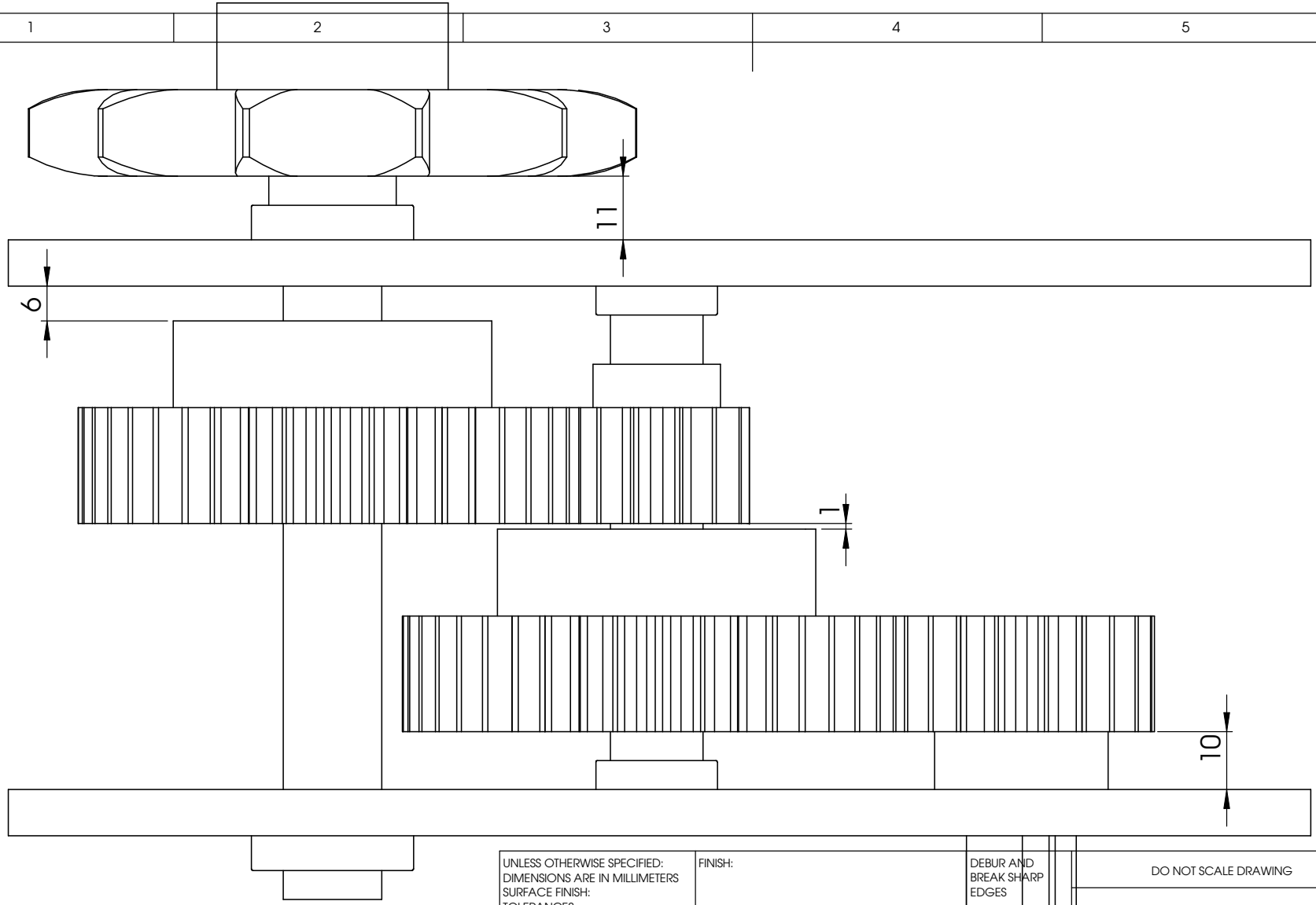
D

Apêndice D - desenhos técnicos das peças do redutor do motor de tração

Installation dimensions



Shaft - 19mm diameter x 40mm long
with ISO keyway & centre
hole tapped M8x1.25



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION

	NAME	SIGNATURE	DATE
DRAWN			
CHK'D			
APPV'D			
MFG			
Q.A			

MATERIAL:

WEIGHT:

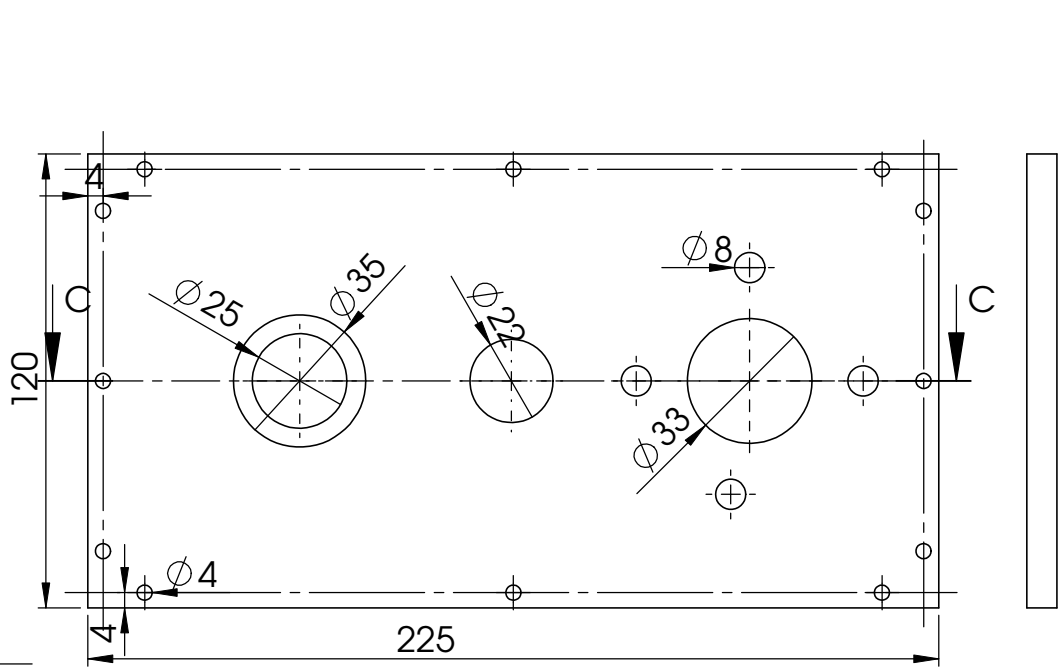
TITLE:

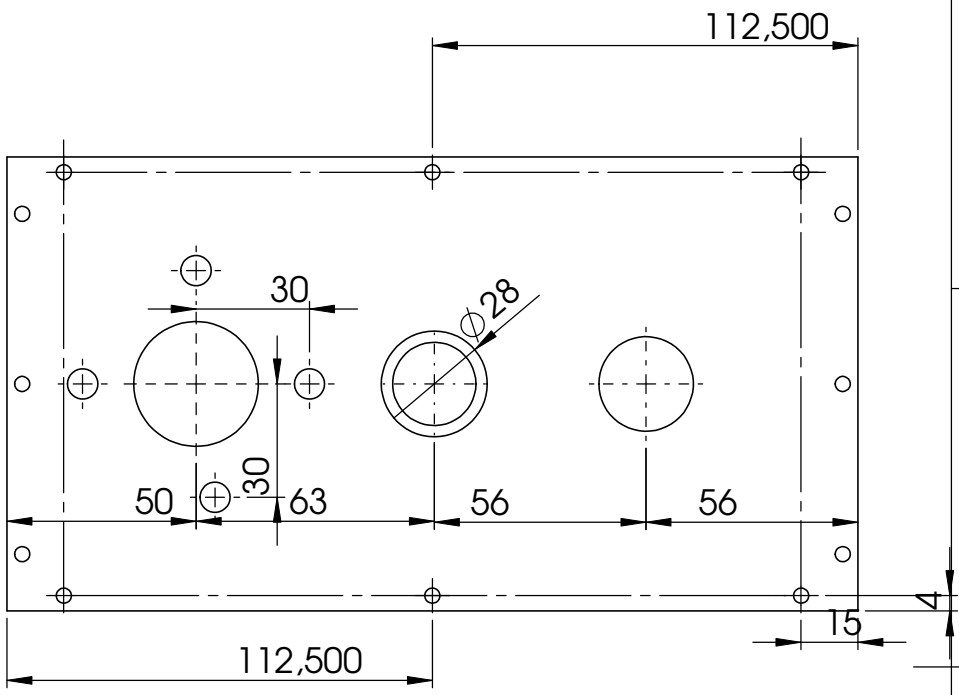
DWG NO. montagem

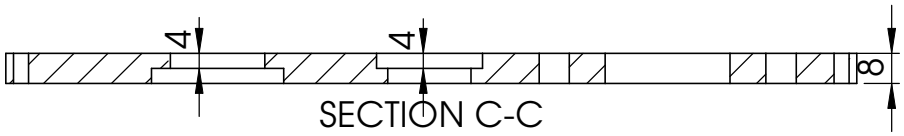
A4

SCALE:1:5

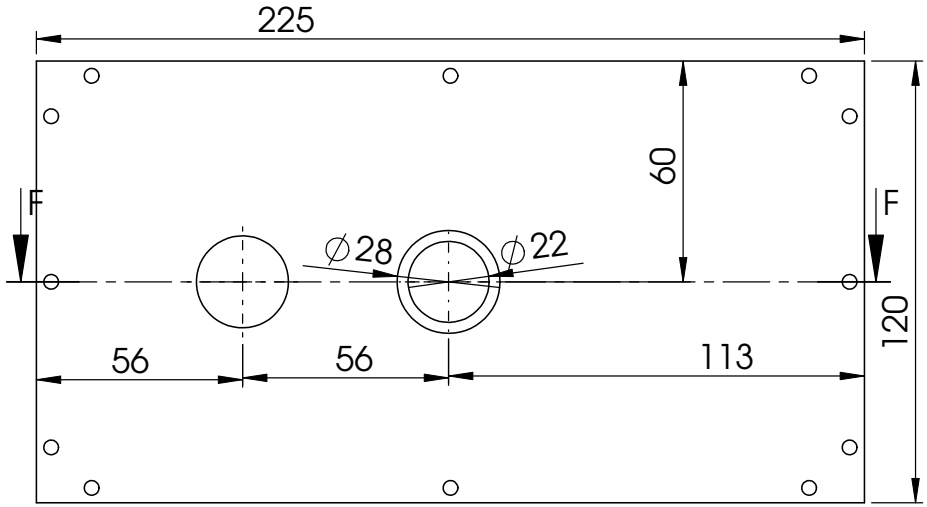
SHEET 1 OF 1

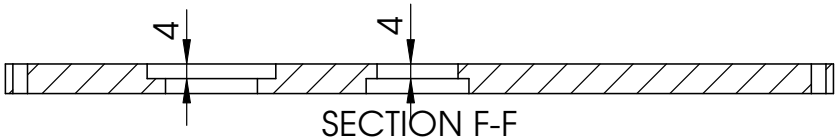




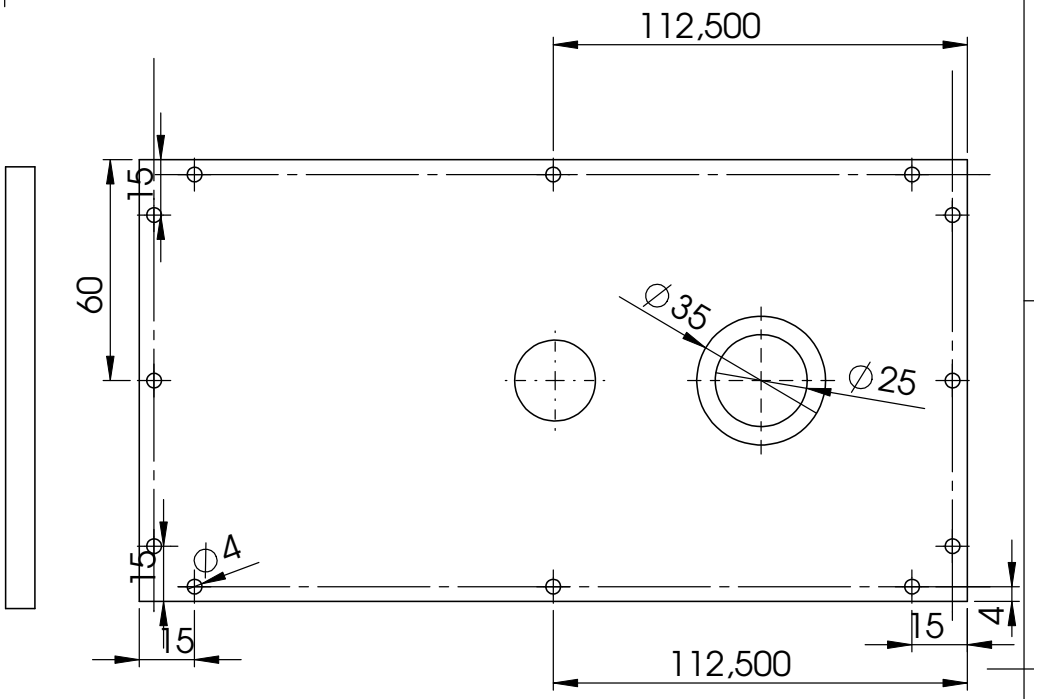


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE	TITLE:		
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. chapa motor	A4
				WEIGHT:	SCALE:1:2	SHEET 1 OF 1

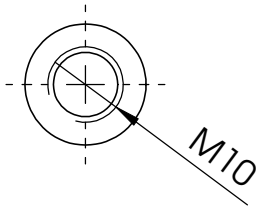


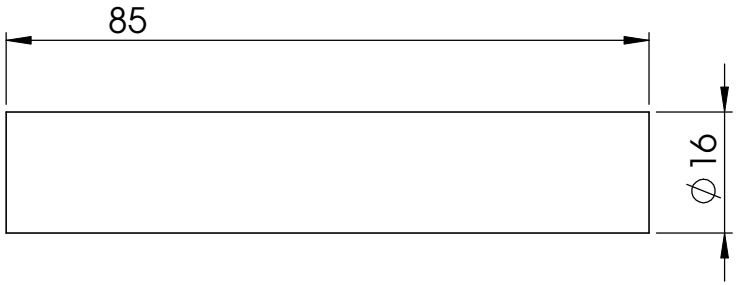


SECTION F-F

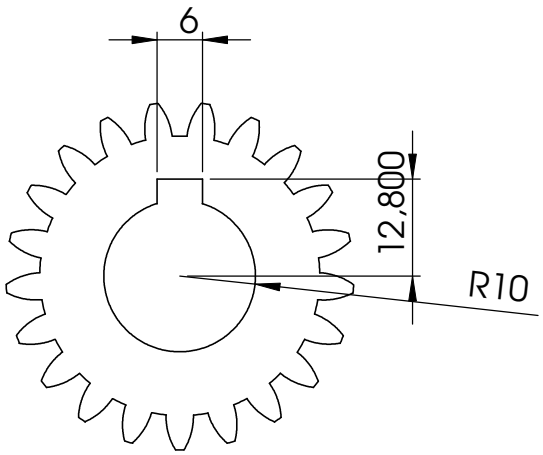


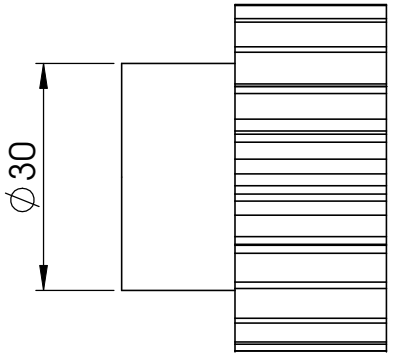
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE			TITLE:
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. chapa corrente / A4
				WEIGHT:		SCALE:1:2 / SHEET 1 OF 1

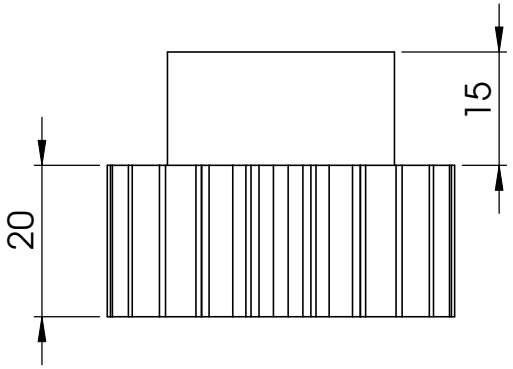




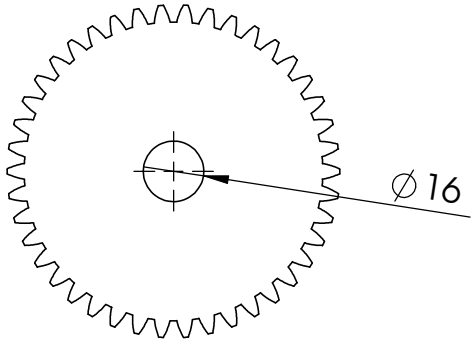
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE			TITLE: espaçador chapa
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. A4
				WEIGHT:		SCALE:1:1 / SHEET 1 OF 1

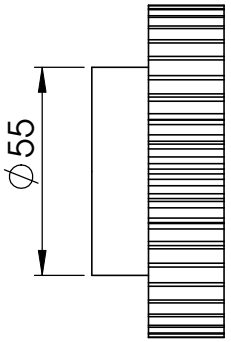


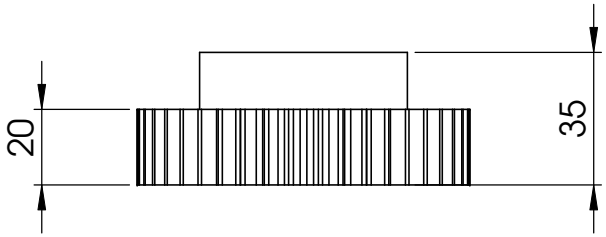




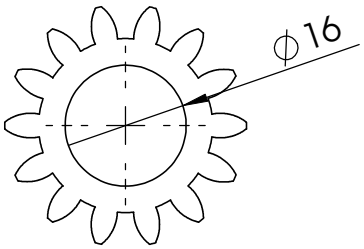
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE			TITLE:
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. roda 21	A4
				WEIGHT:	SCALE:1:1	SHEET 1 OF 1

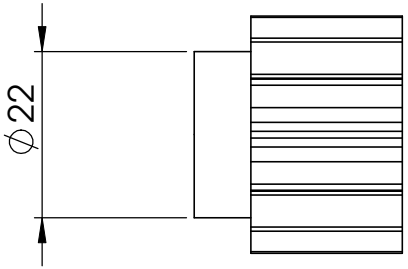


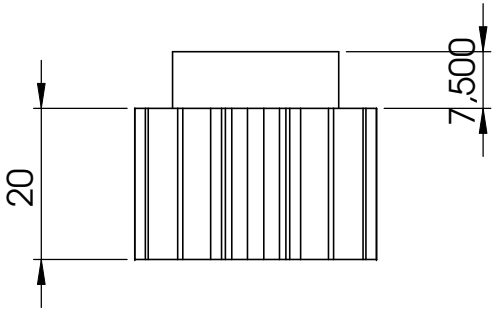




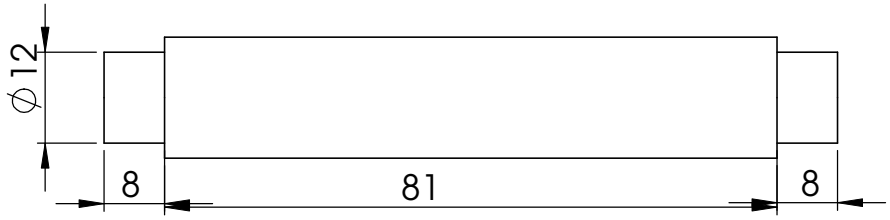
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE		TITLE:	
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. roda 42 16	A4
				WEIGHT:	SCALE:1:2	SHEET 1 OF 1

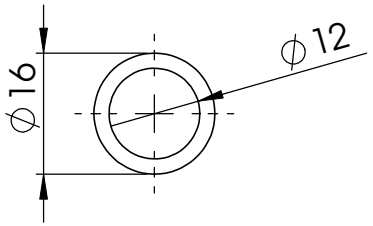


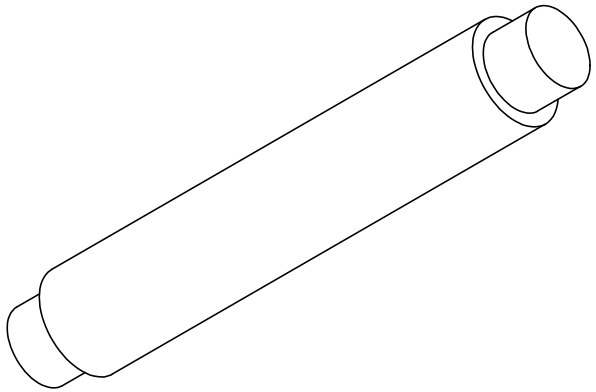




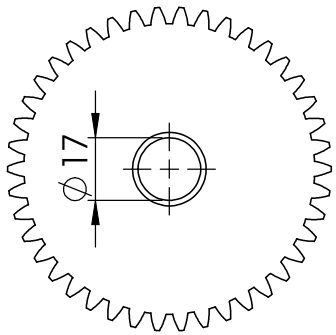
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. roda 14 16	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1

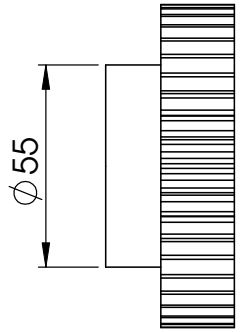


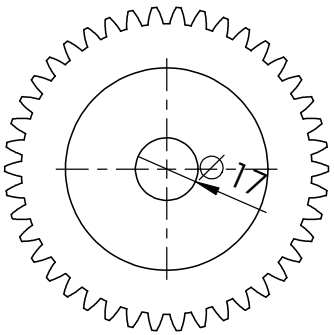


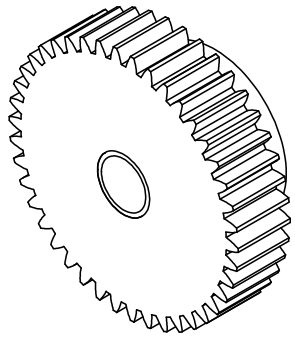


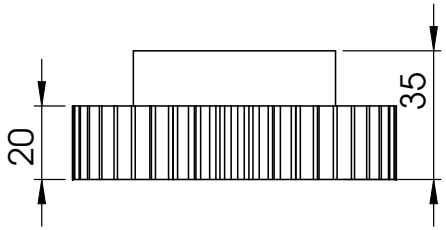
UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBUR AND BREAK SHARP EDGES		DO NOT SCALE DRAWING	REVISION
	NAME	SIGNATURE	DATE			TITLE:	
DRAWN							
CHK'D							
APPV'D							
MFG							
Q.A				MATERIAL:		DWG NO. veio16.12	A4
				WEIGHT:		SCALE:1:1	SHEET 1 OF 1



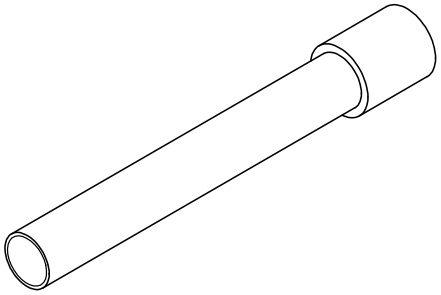


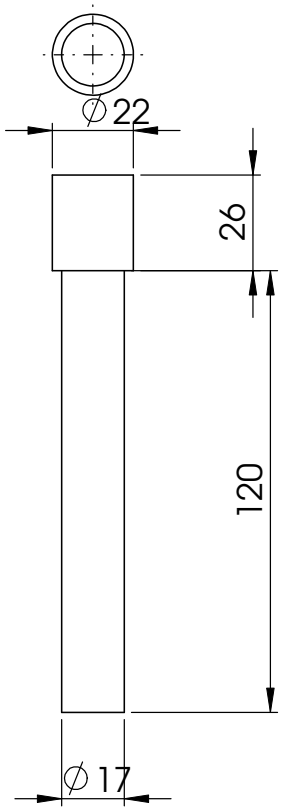


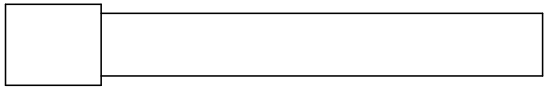


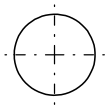


UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE			TITLE:
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. roda 42 20 / A4
				WEIGHT:		SCALE:1:2 / SHEET 1 OF 1

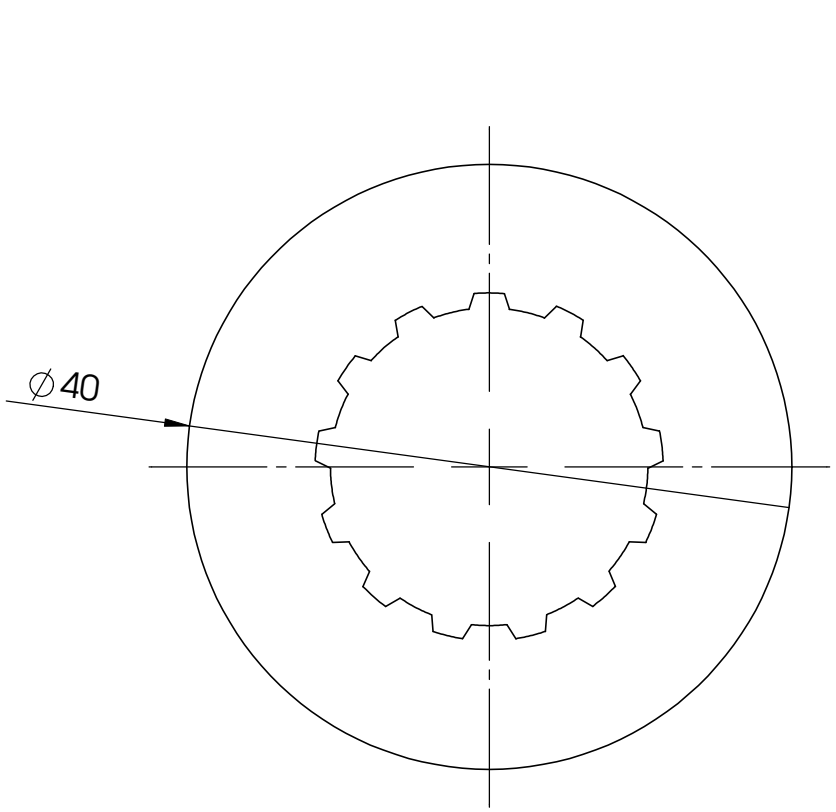


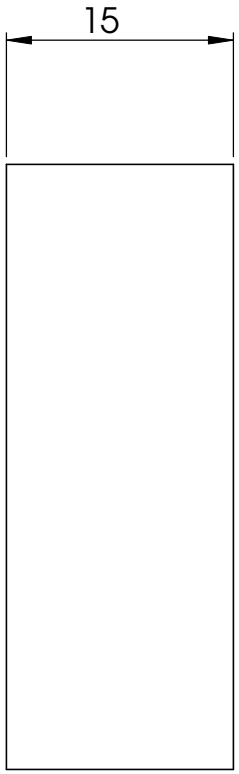






UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:			DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING / REVISION
	NAME	SIGNATURE	DATE			TITLE:
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:		DWG NO. 20.23 / A4
				WEIGHT:		SCALE:1:2 / SHEET 1 OF 1





UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:		DEBUR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION	
	NAME	SIGNATURE	DATE			TITLE:
DRAWN						
CHK'D						
APPV'D						
MFG						
Q.A				MATERIAL:	DWG NO. Bolacha	A4
				WEIGHT:	SCALE:2:1	SHEET 1 OF 1

Os desenhos deste anexo são uma especificação aproximada do redutor *à priori*, com as limitações inerentes à falta de experiência do projetista. Não contemplam alguns pormenores técnicos avançados, como tolerâncias nos encaixes, materiais a usar, e mecanismos de fixação. Não substituem o parecer de técnicos especializados.

Como tal, o fabricante fez algumas modificações, discutidas na errata

Informações sobre o fabricante

Vitor Ferreira & Filhos, Lda

Rua Particular à Rua Arco do Carvalhão, Letras J.F.C, 1070 Lisboa

Telefone: 213884764

www.mestredosmotores.com

Errata

Errata referente aos desenhos das peças do redutor do motor de tração

talking point (16) **To do** (17)

Desenho	Onde se lê/vê	Deve lêr-se/vêr-se
espaçador chapa	85	87
roda 21	R10	R9.5
roda 21	12.800	12.300
roda 21		Chave (paralelipípedo quadrangular com 6 mm de lado, cantos arredondados e 35 mm de comprimento) para prender engrenagem de 21 dentes ao veio do motor, de acordo com a norma ISO/R773 [16].
20.23	Face do veio ⌀22 liso	Face do veio ⌀22 liso até 11mm após o ⌀17. Daí até ao topo, veio maquinado para encaixe no buraco da bolacha do desenho "bolacha".**talking point** (18)
20.23	Topo do veio ⌀22 liso	Topo do veio ⌀22 com rosca M8 concêntrica.
20.23	120	115, medidos a partir do veio ⌀22. Ranhura para freio após a medida.**To do** (19)
bolacha		Um dos topos do cilindro tapado, com um furo M8 concêntrico.
chapa corrente	12 furos de 4 mm	4 furos M10 próximos dos cantos da placa.
chapa motor	12 furos de 4 mm	4 furos M10 próximos dos cantos da placa, à mesma distância dos da errata do desenho "chapa corrente".
montagem	Veio do motor	Espaçador cilíndrico com furo concêntrico de 19 mm e cerca de 3 mm de largura, montado no veio, antes da engrenagem.

To do...

- □ 1 (p. A): devo pôr a foto da academia militar? As regras do ist não preveem isso
- □ 2 (p. A): foto com baterias e sem carro
- □ 3 (p. A): devi pôr Dr. também?
- □ 4 (p. xiii): Como escrever o acronimo ROVIM
- □ 5 (p. 3): **rever** devo dizer os critérios de seleção tecnológica antes de descrever e avaliar as tecnologias? Isso já está +- feito na secção acima, mas o chan refere quais os desafios que a tecnologia tem que responder.
- ☑ 6 (p. 5): É dada especial relevância à escolha de um propulsor usado em veículos comerciais desta dimensão (desde que cumpra os requisitos desta aplicação). Isto aumenta o grau de confiança na validade da configuração usada e faz com que haja mais documentação acerca da sua implementação prática em projetos anteriores.
- □ 7 (p. 8): **rever** o que dizer aqui? Apresentar o ziegler nichols? Mas ele só foi usado depois de ter falhado a modulação e identificação...
- □ 8 (p. 8): **rever** como enquadrar esta secção no capítulo: isto não é propriamente uma revisão da literatura, é mais uma análise de alto nível do problema
- □ 9 (p. 8): referir aqui os motores e acopladores a usar e controladores? e o sensor?
- □ 10 (p. 8): referir que não espero grande impacto na autonomia deste módulo?
- □ 11 (p. 9): o que dizer aqui mais? Incorporar as irregularidades do terreno?
- □ 12 (p. 18): colocar bibliografia em portugues
- □ 13 (p. A-2): mostrar restantes ficheiros de código
- □ 14 (p. A-2): **Fix** a listagem do código ocupa atualmente 70 páginas
- □ 15 (p. C-3): lista de componentes do sensor de velocidade
- □ 16 (p. D-14): **talking point** Pq não alterei os desenhos em vez de fazer uma errata? Pq alguns parâmetros foram mudados pelo torneiro durante a manufatura (após o desenho da caixa), e documentá-los implicaria desmontar parcialmente ou na totalidade a caixa...
- □ 17 (p. D-14): fazer as correções na errata e alterar o nome dos desenhos
- □ 18 (p. D-14): **talking point** com a largura do carreto, a bolacha não entra totalmente no veio.
- □ 19 (p. D-14): confirmar