RODRIGO PÓVOA

APLICAÇÃO DO PROTOCOLO "KW2000" PARA LEITURA DE DADOS DISPONÍVEIS NO MÓDULO DE CONTROLE DO MOTOR DE UM VEÍCULO POPULAR

Dissertação Apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Automotiva

São Paulo
2007

RODRIGO PÓVOA

PARA CÓPIAS, CONSULTAR A EDIÇÃO REVISADA :

FMP- 124

Ed. rev.

APLICAÇÃO DO PROTOCOLO "KW2000" PARA LEITURA DE DADOS DISPONÍVEIS NO MÓDULO DE CONTROLE DO MOTOR DE UM VEÍCULO POPULAR

Dissertação Apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Engenharia Automotiva

Área de Concentração:
Engenharia Automotiva

Orientador:
Prof. Doutor Lucas Antonio Moscato
Co-orientador:
Prof. Doutor Jun Okamoto Jr.

São Paulo
2007

DEDICATÓRIA

Primeiramente a Deus por permitir a vida humana na Terra mesmo frente a tanta calamidade causada pelo homem, à minha família por incentivar-me a lutar na vida e, em especial, à minha esposa por dividir comigo o jugo desta luta.

RESUMO

Atualmente a programação das ferramentas de diagnóstico eletrônico, utilizadas por oficinas e rede de concessionárias de algumas montadoras de veículos do Brasil, é executada por mão-de-obra internacional com elevado custo, impactando negativamente o custo estrutural das empresas nacionais que dependem deste serviço. Desta forma, surgiu o interesse na nacionalização desta programação. A proposta de pesquisa deste trabalho teve como objetivo a familiarização com o protocolo de diagnóstico "KW2000" ("Keyword 2000"), que atualmente é classificado como o principal protocolo utilizado para leitura de informações dos módulos eletrônicos de controle de dispositivos em veículos nacionais, através de ferramentas de diagnóstico dedicadas ou genéricas, utilizadas por toda a rede de concessionárias e oficinas do país. O projeto foi formado por duas partes principais, a saber, pesquisa dos requerimentos do protocolo por meio de consulta às normas internacionais que padronizam a aplicação deste protocolo e desenvolvimento de uma interface de software através do "Delphi" (linguagem de computação baseada em Pascal), a qual extraiu do módulo de controle do motor de um veículo popular as informações existentes relacionadas ao funcionamento do motor e identificação do próprio módulo, atualizando-as continuamente na tela de um computador que recebe e solicita as informações através da porta serial.

Palavras-Chave: Protocolos de Comunicação, Desenvolvimento de Software, Veículos Populares

ABSTRACT

Nowadays the programming of the electronic diagnosis tool, used by workshops and dealers of some Brazilian automobile companies, is performed by foreign labor with high cost, negatively impacting the structural cost of national companies which depend on this kind of service. This way, it encourages the interest in this programming nationalization. The research proposal of this work was getting familiarized with the diagnostic protocol "KW2000" (Keyword 2000) which, nowadays, is classified as the main protocol used for information exchange between the vehicle control module and the diagnosis tool in Brazilian market. The project was built in two main parts. The first one consists in a protocol requirements research, consulting the international standardizations which define "KW2000" protocol. The second one consists in a software interface development based on "Delphi" (computer language derived from Pascal), which get information from an economy class vehicle engine control module regarding the engine behavior and the controller identification. This data is updated continuously and shown on the computer screen which receives and requests such information through serial communication port.

Keywords: Communication Protocols, Software Development, Economy Class Vehicles.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 Introdução

O grande interesse na familiarização com o protocolo "KW2000", baseia-se na possibilidade de nacionalizar a programação de ferramentas de diagnóstico eletrônico, utilizadas por oficinas e rede de concessionárias.

Atualmente algumas empresas do ramo automobilístico pagam pela realização do desenvolvimento do software de suas ferramentas de diagnóstico em território estrangeiro, a partir de descritivos funcionais adquiridos junto aos fornecedores dos módulos de controle dos dispositivos dos veículos que, em muitas oportunidades, estão localizados em nosso próprio país. Esta prática acarreta um grande impacto no custo estrutural destas empresas (indicador financeiro considerado de importância vital para as empresas do ramo hoje em dia), já que o custo da mão-de-obra de países europeus ou norte-americanos (detentores do conhecimento) são excessivamente superiores ao custo da mão-de-obra nacional. Aliado a isto, ainda há uma despesa adicional relacionada à taxa cobrada pelo governo federal devido à importação de serviços, a qual está hoje em 49,73% (Fonte: Departamento de Compras da General Motors do Brasil).

Uma outra desvantagem está no tocante à flexibilização e agilidade no processo de atualização do software da ferramenta visando cobrir lançamentos de veículos ou eventuais "*bugs*" em programas desenvolvidos anteriormente, uma vez que os autores destas atualizações encontram-se em outros países.

Não há interesse de se criar uma ferramenta de diagnóstico como fruto deste projeto. O interesse é realmente a familiarização com o protocolo para tornar a programação das ferramentas de diagnóstico, que possuem linguagem própria, mais acessível. Para tanto, será explorado o conceito do protocolo "KW2000", a fim de compreender como o protocolo funciona e, desenvolver-se-á uma aplicação em "Delphi" para visualizar o comportamento do protocolo em atividade.

A exploração do conceito do protocolo será realizada, basicamente, através da aplicação da metodologia de levantamento de informações da ISO 14230 (que define o protocolo "KW2000"), bem como das informações contidas no descritivo funcional para diagnóstico do módulo de controle do motor de um veículo popular. Adicionalmente, outras literaturas serão

analisadas com o objetivo de enriquecer o detalhamento das informações referentes ao protocolo.

Sobre a aplicação em "Delphi" para análise do comportamento do protocolo, o diagrama de blocos a seguir expõe a forma de conexão de uma ferramenta de diagnóstico eletrônico genérica a um veículo:

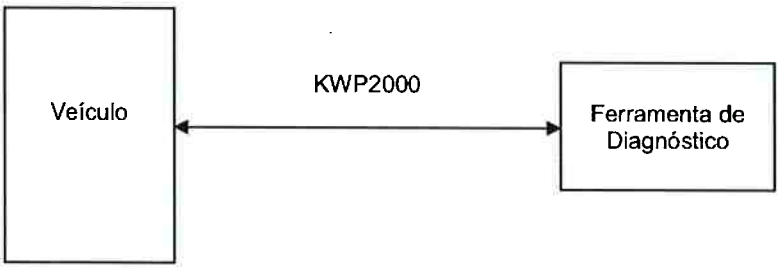


Figura 1 - Conexão da Ferramenta de Diagnóstico

A proposta para familiarização com o protocolo "KWP2000" pode ser observada através do seguinte diagrama de blocos:

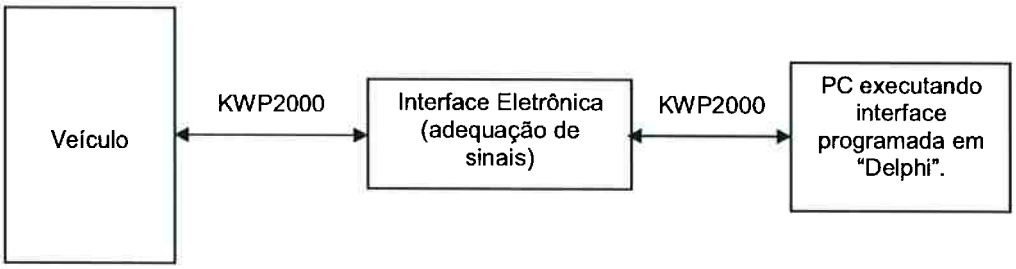


Figura 2 - Proposta de conexão para aplicação em "Delphi"

Relacionando em tópicos, o projeto consiste em:

- Levantamento das informações relacionadas ao protocolo *Keyword* 2000;
- Obtenção de uma Interface Eletrônica (Hardware) para adequação dos sinais elétricos;
- Programação em linguagem "Delphi" para elaboração de uma interface, com a finalidade de exibir os dados que serão colhidos, através da porta serial RS-232.

A seguir são apresentadas as informações referentes ao protocolo.

2 O Protocolo *Keyword* 2000

Com a explosão eletrônica no ramo automobilístico no final dos anos 80 e começo dos anos 90, cada montadora acabou recorrendo a protocolos de comunicação próprios para obtenção de dados de diagnóstico eletrônico em veículos automotores. Desta forma, existiam inúmeras vertentes de soluções para cada problema encontrado relacionado a esta atividade.

Em meados dos anos 90, várias empresas do ramo automotivo se juntaram para criar um protocolo de comunicação padronizado, que posteriormente foi batizado de "*Keyword Protocol* 2000". A seguir são apresentadas as empresas que participaram deste processo:

- Adam Opel AG
- AISIN AW CO., Limited Japan
- Audi AG / Volkswagen AG
- BMW AG
- Daimler-Benz AG
- debis Systemhaus GmbH
- DELCO Electronics Europe
- DSA Daten und Systemtechnik GmbH
- ETAS GmbH & Co. KG
- FEV Motorentechnik GmbH & Co. KG
- GenRad Europe Ltd.
- GM Europe GmbH Service Technology Group Int'l Operations
- Hella KG
- Isuzu Motors Ltd.
- Kelsey-Hayes
- LucasVarity
- MAN Nutzfahrzeuge AG
- Mecel AB
- Robert Bosch GmbH
- Saab Automobile AB
- Siemens AG
- Softing GmbH

- VDO Adolf Schindling AG

Hoje o protocolo "KW2000" é amplamente utilizado no desenvolvimento de módulos eletrônicos e ferramentas de diagnóstico para o setor automobilístico, o que incentivou o aprofundamento da pesquisa do protocolo através deste trabalho.

A seguir serão apresentadas as especificações deste protocolo.

3 Especificações

3.1 Topologia física

O protocolo "KW2000" utiliza o conceito *bus*, tendo como estrutura duas linhas seriais para transmissão de dados: Linha K, que é utilizada para comunicação e inicialização e, Linha L, que é opcional e utilizada apenas para inicialização. A figura a seguir ilustra esta estrutura.

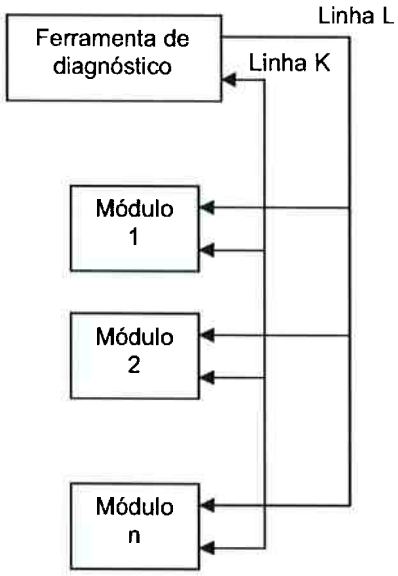


Figura 3 - Topologia

Uma outra estrutura possível é a conexão nó-a-nó, onde a ferramenta de diagnóstico é conectada somente a um módulo eletrônico e também utiliza o conceito *bus*.

3.2 Estrutura da Mensagem

A mensagem de dados no protocolo "KW2000" é formada por três partes:
- cabeçalho
- *bytes* de dados
- verificação (*Checksum*)

A figura a seguir mostra a estrutura da mensagem.

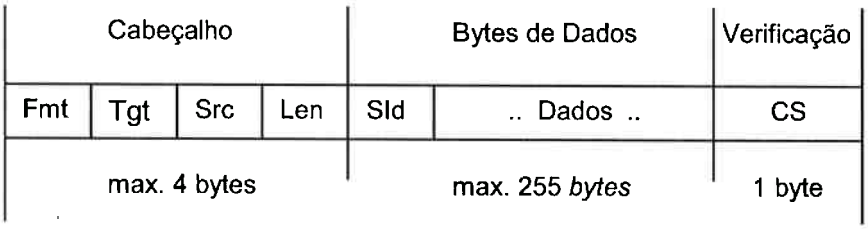


Figura 4 - Estrutura da mensagem

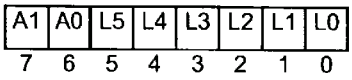
Obs.: Os *bytes* Tgt, Src e Len são opcionais, dependendo do *Format Byte* (Fmt) – veja 3.2.1.1.

3.2.1 Cabeçalho

O cabeçalho é formado por no máximo 4 *bytes*, conforme ilustrado na figura 4. O *Format Byte* (Fmt) contém informações sobre o formato da mensagem. O *Target Byte* (Tgt) e o *Source Byte* (Src) são opcionais para uso em conexões de múltiplos nós. O *Additional Lenght Byte* (Len) também é opcional e especifica mensagens de até 255 *bytes*.

3.2.1.1 *Format Byte* (Fmt)

O *Format Byte* contém 6 *bits* que podem ser utilizados para informar quantos *bytes* de dados, incluindo o SId (*Service Identification*) serão enviados na mensagem (desde que este número seja menor que 64 - Tabela 2) e 2 *bits* com informação sobre o endereçamento (existência e modo).



Os *bits* A1 e A0 podem assumir os seguintes valores:

A1	A0	Significado
0	0	Sem informação de endereçamento
0	1	Modo de exceção (CARB)
1	0	Com informação física de endereçamento
1	1	Com informação funcional de endereçamento

Tabela 1 - Forma do cabeçalho da mensagem

Obs.: O modo de exceção CARB não é objeto de estudo deste trabalho. Informações sobre este modo podem ser obtidas através das normas ISO 9141-2 e SAE J1979.

3.2.1.2 *Target Byte* (Tgt)

Este *byte* especifica o endereço do objetivo da mensagem e é sempre usado com o *Source Byte*. Quando usado na mensagem de solicitação, ou seja, da ferramenta de diagnóstico para os módulos eletrônicos, ele pode indicar endereçamento físico ou funcional. Já a mensagem de retorno dos módulos para a ferramenta de diagnóstico, deve sempre indicar o endereçamento físico. Somente se faz necessário o seu uso quando se trata de uma estrutura com múltiplos nós.

Obs.: Para saber o correto endereço a ser utilizado, pode-se consultar as normas ISO 14230-2 e SAE J2178-1.

3.2.1.3 *Source Byte* (Src)

Este *byte* é utilizado em conjunto com o *Target Byte* e somente é necessário para estruturas de múltiplos nós, indicando o endereço do dispositivo que está transmitindo a mensagem. Deve sempre ser endereçamento físico, que é especificado nas normas ISO 14230-2 e SAE J2178-1.

3.2.1.4 *Additional Lenght Byte* (Len)

Este *byte* é utilizado quando os *bits* de tamanho do *Format Byte* estão zerados (L0 a L5), conforme mostrado na Tabela 2. Em geral sua utilização ocorre quando os *bytes* de dados (incluindo o SId) são superiores a 63 *bytes*, limitados a 255. Também pode ser utilizado para mensagens com um número de *bytes* de dados inferior a 64, porém o *Format Byte* é capaz de indicar este tamanho, tornando o *Additional Lenght Byte* desnecessário.

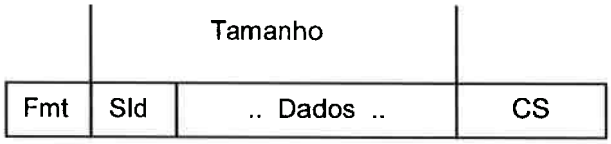
Tamanho	Tamanho indicado no	
	Format Byte	***Additional Lenght Byte***
< 64	XX00 0000	Presente
< 64	XXLL LLLL	Não presente
≥ 64	XX00 0000	Presente

Tabela 2 - Presença do *Additional Lenght Byte*

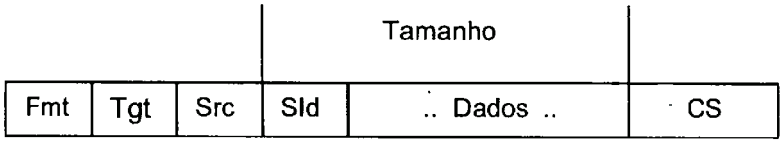
Obs.: XX: 2 *bits* com informação sobre o endereçamento
LL LLLL: 6 *bits* com informação sobre o tamanho do *byte* de dados

3.2.1.5 Tipos de mensagens

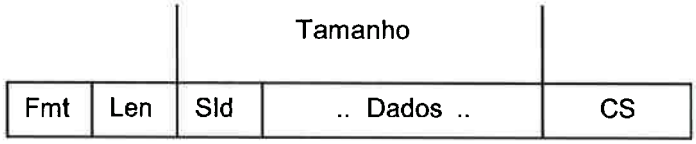
Com estas definições é possível definir 4 formas de mensagens, exibidas na figura a seguir:



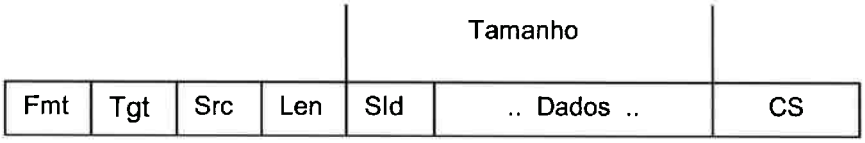
a) Cabeçalho com informação de modo de endereço, sem o *Additional Lenght Byte*



b) Cabeçalho com informação de endereço, sem o *Additional Lenght Byte*



c) Cabeçalho sem informação de endereço, com o *Additional Lenght Byte*



d) Cabeçalho com informação de endereço, com o *Additional Lenght Byte*

Fmt – *Format Byte*

Tgt – *Target Byte* (opcional)

Src – *Source Byte* (opcional)

Len – *Additional Lenght Byte* (opcional)

SId – *Service Identification*

Dados – Dados da Mensagem

CS – *Checksum*

Figura 5 - Tipos de mensagem

3.2.2 *Bytes* de Dados

Este campo da mensagem pode conter até 63 *bytes* ou até 255 *bytes*, dependendo da definição do tamanho da mensagem no cabeçalho (veja 3.2.1 Cabeçalho). Seu primeiro *byte* é o *Service Identification* (SId), que pode ser seguido por parâmetros ou dados, dependendo do serviço selecionado. Estes *bytes* são definidos na ISO 14230-3 no tocante a serviços de diagnóstico.

3.2.3 Verificação (*Checksum*)

O *byte* de verificação existente no fim da mensagem pode ser definido como uma simples somatória de todos os *bytes* da mensagem, excluindo ele próprio e limitando-se a 2 caracteres em hexadecimal.

Como exemplo, suponha que foram recebidos 6 *bytes* com os seguintes conteúdos (em hexadecimal):

- 80h / 11h / F1h / 02h / 21h / 01h

O valor do *Checksum* é dado por: 80h+11h+F1h+02h+21h+01h = 1A6h, que limitado a 2 caracteres é igual a A6.

3.2.4 Intervalo de Tempo

3.2.4.1 Parâmetros

Durante operação normal, os parâmetros de intervalo de tempo se comportam como ilustrado na figura a seguir:

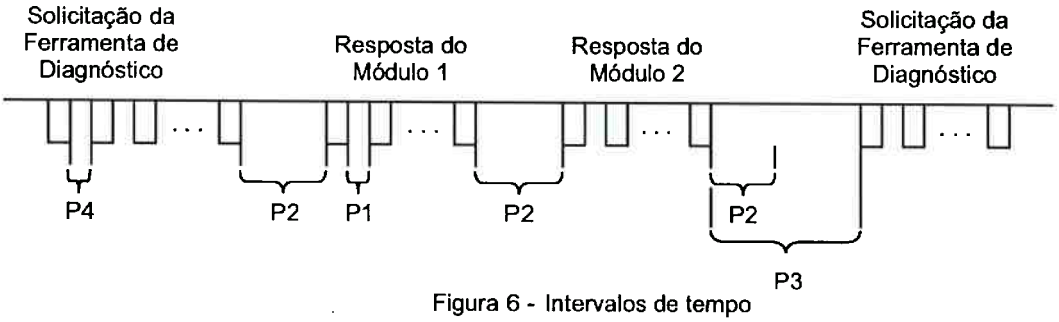


Figura 6 - Intervalos de tempo

Intervalo	Descrição
P1	Intervalo de tempo entre os *bytes* da resposta do módulo
P2	Intervalo de tempo entre a solicitação da ferramenta de diagnóstico e a resposta do módulo ou o tempo entre duas respostas de módulos diferentes
P3	Intervalo de tempo entre a última resposta dos módulos e o início de uma nova solicitação da ferramenta de diagnóstico (Exceção – 4.2.4.2)
P4	Intervalo de tempo entre os *bytes* da solicitação da ferramenta de diagnóstico

Tabela 3 - Descrição dos intervalos de tempo

Existem duas configurações padrões para os intervalos de tempo:

1 – Configuração para comunicação com endereçamento físico e funcional convencional, onde são necessários tempos longos para permitir inclusão de gerenciamento de barramento;

2 – Configuração restrita para endereçamento físico, permitindo comunicação mais rápida.

A ferramenta de diagnóstico é informada quanto à capacidade de um módulo através de *Key Bytes* (veja 3.3.2.4.1 *Key Bytes*). Os parâmetros relacionados aos intervalos de tempo podem ser modificados com o serviço de comunicação *"AccessTimingParameter"* (veja 3.3.4 Serviço *"AccessTimingParameter"*).

É importante notar algumas limitações quanto ao uso dos parâmetros de intervalos de tempo, como seguem:

P3min > P4min
Pimin > Pimax, onde i = 1, 2, 3 ou 4

Para evitar confusões quanto à detecção de fim de mensagem por estouro do tempo, as seguintes regras devem ser respeitadas:

P2min > P4max

P2min > P1max

Como pode ser observado, a escolha de parâmetros apropriados é primordial para o bom funcionamento da comunicação entre a ferramenta de diagnóstico e os módulos eletrônicos. É muito importante respeitar as limitações dos módulos e da ferramenta de diagnóstico, definindo os parâmetros de intervalo de tempo conforme as características mais críticas (piores casos) dos dispositivos que compõem a rede de comunicação. As tabelas a seguir apresentam os valores padrões para estes parâmetros, apontando os limites e a resolução que devem ser usados para configurar um outro valor através do serviço de comunicação *"AccessTimingParameter"*:

Parâmetro	Valores Mínimos			Valores Máximos		
	Limite Inferior	Padrão	Resolução	Padrão	Limite Superior	Resolução
P1	0 (ms)	0 (ms)	-	20 (ms)	20 (ms)	-
P2	0 (ms)	25 (ms)	0,5 (ms)	50 (ms)	Veja tabela 4	
P3	0 (ms)	55 (ms)	0,5 (ms)	5000 (ms)	∞	250 (ms)
P4	0 (ms)	5 (ms)	0,5 (ms)	20 (ms)	20 (ms)	-

Tabela 4 - Configuração Normal dos Parâmetros de Intervalo de Tempo

Valor Hex.	Resolução	Valor Máximo	Método para o Cálculo do Valor Máximo
01 a F0	25 (ms)	25 a 6000 (ms)	(Valor Hex.) x (Resolução)
F1	Veja a Coluna "Método para o Cálculo do Valor Máximo"	6400 (ms)	(Parte Baixa do Valor Hex.) x 256 x 25 Exemplo: Valor Hex. = FA Parte Baixa = A $(A)_{16} = 10$ 10 x 256 x 25 = 64000
F2		12800 (ms)	
F3		19200 (ms)	
F4		25600 (ms)	
F5		32000 (ms)	
F6		38400 (ms)	
F7		44800 (ms)	
F8		51200 (ms)	
F9		57600 (ms)	
FA		64000 (ms)	
FB		70400 (ms)	
FC		76800 (ms)	
FD		83200 (ms)	
FE		83600 (ms)	
FF	-	∞	Não Aplicável
O valor do parâmetro deve sempre ser um valor de *byte* individual no serviço *"AccessTimingParameter"*. As modificações dos intervalos devem ser ativadas através da implementação do serviço *"AccessTimingParameter"*.			

Tabela 5 - Cálculo do Parâmetro P2max

3.2.4.2 Exceções

O intervalo de tempo definido pelo parâmetro P2 pode ser estendido para possibilitar que o servidor responda a uma solicitação em um tempo superior ao limitado por P2. Esta exceção somente é permitida quando ocorre uma ou várias mensagens de resposta negativa, cujo código é 78h ("*Request Correctly Received - Response Pending*"), proveniente do servidor. Tal código de resposta pode ser usado somente nos casos em que o servidor não conseguir enviar uma resposta negativa ou positiva, relativa à mensagem recebida, durante o intervalo P2. Este processo caracteriza a incapacidade do servidor de responder ao cliente no intervalo P2, o que deve fazer com que este parâmetro seja ajustado com o mesmo valor do parâmetro P3, permitindo nova tentativa por parte do servidor.
Assim que o servidor finalizar a tarefa solicitada pelo cliente, ele deve enviar um código de resposta negativa ou positiva (diferente do código 78h), baseado na mensagem recebida. Quando o cliente recebe a primeira mensagem diferente do código 78h, tanto o servidor como o cliente devem ajustar o parâmetro P2 para o valor anterior à primeira resposta negativa 78h recebida, restaurando o intervalo de tempo padrão.

3.3 Estrutura da Mensagem

3.3.1 Geral

Alguns serviços são necessários para manter e estabelecer a comunicação entre os componentes da rede. Tais serviços, semelhante aos de diagnóstico, são formalizados na norma ISO14229, que explica seu significado e o procedimento de aplicação. Seus parâmetros são classificados como:

- Mandatório = M
- Seleção Possível = S
- Condicional = C
- Opcional do usuário = U

Geralmente, estes serviços não são mandatórios. Somente o serviço *"StartCommunication"* deve ser implementado. Este serviço, assim como o *"AccessTimingParameter"*, é usado para iniciar uma comunicação de diagnóstico. A figura a seguir ilustra o uso do serviço de comunicação.

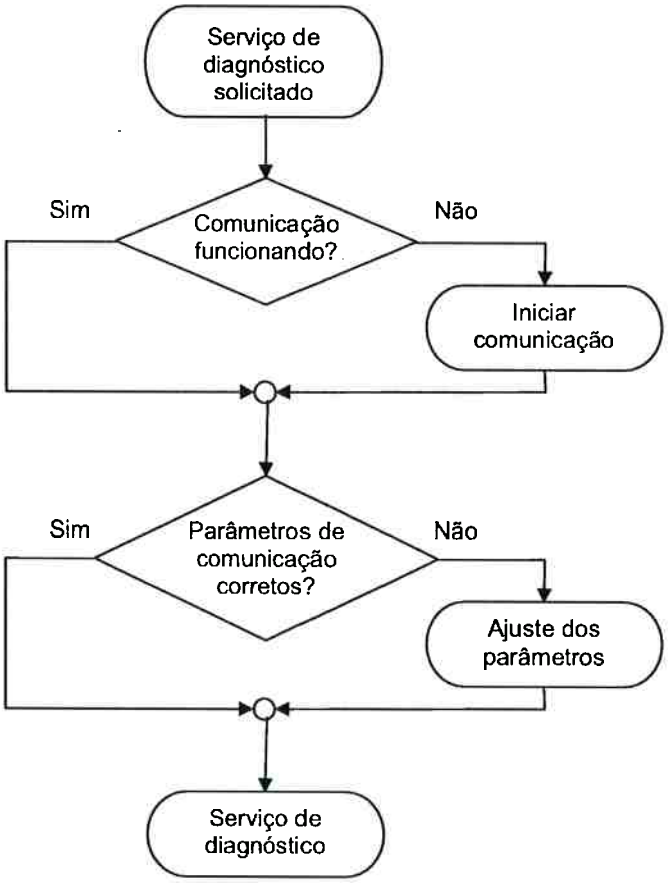


Figura 7 - Uso dos serviços de comunicação

3.3.2 Serviço *"StartCommunication"*

3.3.2.1 Definição do Serviço

A função deste serviço é iniciar a comunicação para a troca de dados de diagnóstico.

3.3.2.2 Tabela de Serviço

A tabela a seguir descreve os parâmetros do serviço supracitado.

Parâmetro	Classificação
"StartCommunication Request"	M
Identificador de Modo de Inicialização	M
Endereço de Inicialização do Objetivo	M
Endereço de Inicialização da Origem	C1
"StartCommunication Positive Response" Endereço do Objetivo Endereço da Origem *Key Bytes*	M C2 C2 M2
1) O caminho da inicialização é determinado pelo identificador do modo de inicialização. 2) C1: Endereço da Inicialização da Origem é adicionado se o Identificador do Modo de Inicialização representa uma Inicialização *Fast* 3) C2: Endereço de Objetivo e Origem são adicionados se a informação do endereço é usada no cabeçalho.	

Tabela 6 - Serviço *"StartCommunication"*

3.3.2.3 Procedimento de Serviço

Uma vez recebida uma indicação inicial de *"StartCommunication"*, o módulo eletrônico deve verificar se a conexão solicitada pode ser inicializada com as condições do momento. Isto feito, o módulo deve executar todas as ações necessárias para iniciar a conexão de comunicação e enviar uma resposta inicial de *"StartCommunication"* com o parâmetro *"Positive Response"* selecionado. Caso a conexão não possa ser inicializada por qualquer motivo, o módulo deve se manter em operação normal (consulte Serviço *"StartCommunication"* na seção 3.3.6 Manuseando Erros).

3.3.2.4 Implementação

O serviço *"StartCommunication"* é utilizado para inicializar uma comunicação na linha K. Existem diferentes possibilidades para fazer isso:

- Inicialização CARB;

- Inicialização *"5-Baud"*;

- Inicialização *Fast*.

A figura a seguir apresenta as três possibilidades e o estado do módulo após cada tipo de inicialização. Terminada a inicialização, os módulos envolvidos ficarão no mesmo estado:

- Todos os parâmetros de comunicação são carregados com os valores padrões de acordo com os *Key Bytes*;

- O módulo fica aguardando a primeira solicitação da ferramenta de diagnóstico por um período equivalente a P3;

- O módulo fica no modo de diagnóstico padrão, tendo sua funcionalidade bem definida.

Existem alguns fatores que são comuns para todos os modos de inicialização:

- Antes de qualquer atividade deve haver um intervalo para garantir que não ocorra conflitos na rede (*"bus-idle time"*);

- A ferramenta de diagnóstico envia um modelo de inicialização;

- Todas as informações que são necessárias para estabelecer a comunicação são parte integrante da resposta do módulo.

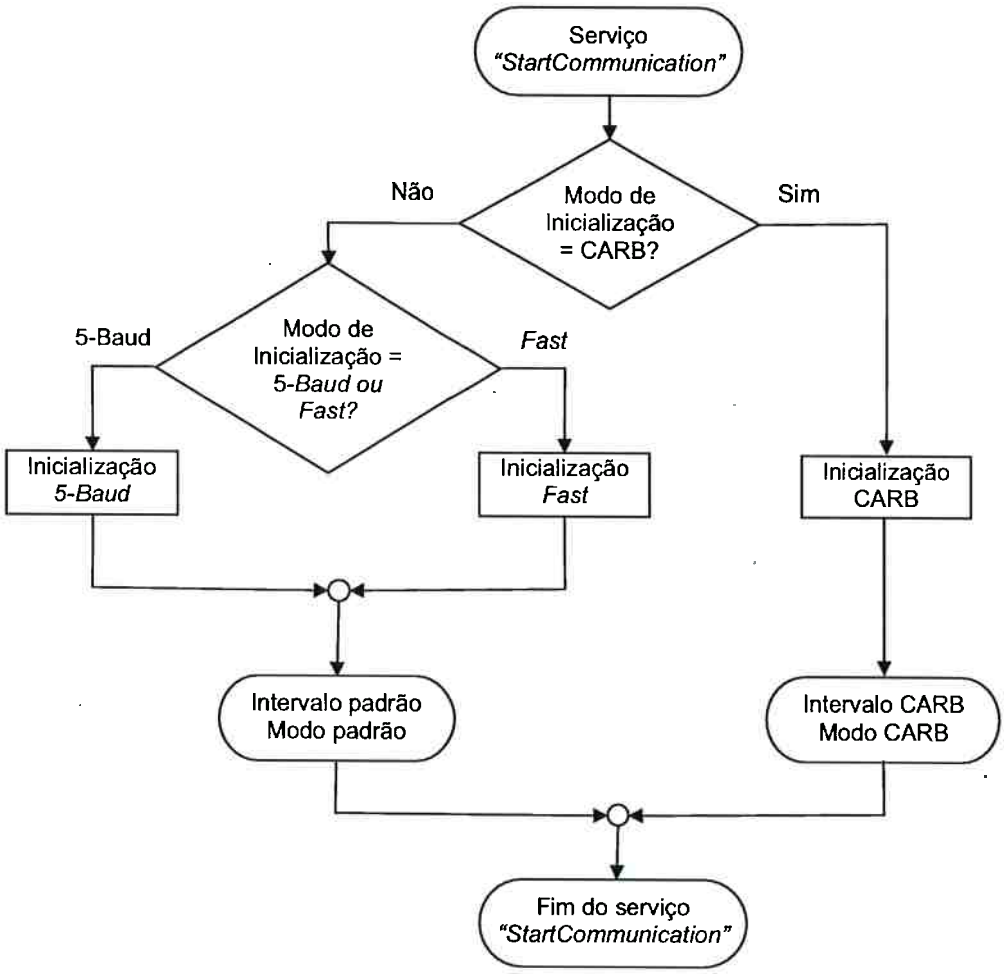
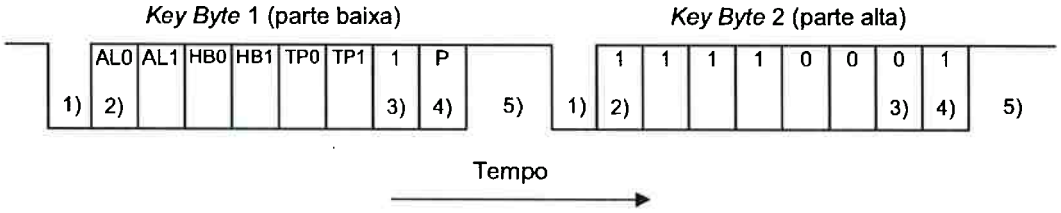


Figura 8 - Modos de Inicialização

3.3.2.4.1 *Key Bytes (KB)*

É através destes *bytes* que um módulo informa a ferramenta de diagnóstico sobre o cabeçalho, intervalo e tamanho da informação. Assim, um módulo não tem que suportar todas as possibilidades destes parâmetros. Basta informar à ferramenta de diagnóstico quais são suportados. A decodificação dos *Key Bytes* é definida na ISO 9141.

A figura a seguir mostra a representação gráfica dos *Key Bytes*. O significado de cada um de seus *bits* está na tabela 7 e os possíveis valores na Tabela 8.



1) *Bit* de início
2) *Bit* menos significativo
3) *Bit* mais significativo
4) *Bit* de equalização
5) *Bit* de paridade

Figura 9 - *Key Bytes*

Bit	**Valor**	
	0	**1**
AL0	Tamanho da Informação não suportada no *Format Byte*,	Tamanho da Informação suportada no *Format Byte*
AL1	*Additional Lenght Byte* não suportado	*Additional Lenght Byte* suportado
HB0	1 *Byte* de cabeçalho não suportado	1 *Byte* de cabeçalho suportado
HB1	Endereços de Objetivo e Origem não suportados	Endereços de Objetivo e Origem suportados
TP0[1)]	Parâmetro de intervalo normal selecionado	Parâmetro de intervalo estendido selecionado
TP1[1)]	Parâmetro de intervalo estendido selecionado	Parâmetro de intervalo normal selecionado
1) Somente TP0, TP1 = 0, 1 ou 1, 0 são permitidos		

Tabela 7 - Significado dos *Bits* nos *Key Bytes*

Key Bytes				Suportado		Intervalo
Binário		Hex.	Dec.[1]	Tamanho da Informação	Tipo de Cabeçalho	
*KB*2	*KB*1					
1000 1111	1101 0000	$8FD0	2000	[2]		
1000 1111	1101 0101	$8FD5	2005	*Format Byte*	Sem informação de endereço	Estendido
1000 1111	1101 0110	$8FD6	2006	*Additional Lenght Byte*		
1000 1111	0101 0111	$8F57	2007	Ambos os Modos		
1000 1111	1101 1001	$8FD9	2009	*Format Byte*	Com informação de endereço de Origem/Objetivo	
1000 1111	1101 1010	$5FDA	2010	*Additional Lenght Byte*		
1000 1111	0101 1011	$8F5B	2011	Ambos os Modos		
1000 1111	0101 1101	$8F5D	2013	*Format Byte*	Ambos os Tipos	
1000 1111	0101 1110	$8F5E	2014	*Additional Lenght Byte*		
1000 1111	1101 1111	$8FDF	2015	Ambos os Modos		
1000 1111	1110 0101	$8FE5	2021	*Format Byte*	Sem informação de endereço	Normal
1000 1111	1110 0110	$8FE6	2022	*Additional Lenght Byte*		
1000 1111	0110 0111	$8F67	2023	Ambos os Modos		
1000 1111	1110 1001	$8FE9	2025	*Format Byte*	Com informação de endereço de Origem/Objetivo	
1000 1111	1110 1010	$8FEA	2026	*Additional Lenght Byte*		
1000 1111	0110 1011	$8F6B	2027	Ambos os Modos		
1000 1111	0110 1101	$8F6D	2029	*Format Byte*	Ambos os Tipos	
1000 1111	0110 1110	$8F6E	2030	*Additional Lenght Byte*		
1000 1111	1110 1111	$8FEF	2031	Ambos os Modos		

1) Para o cálculo do valor decimal, zerar o *bit* de paridade de cada *Key Byte*, multiplicar o *Key Byte* 2 (*KB*2) por 128 (2^7) e somar o *Key Byte* 1 (*KB*1);

2) Com valor decimal de 2000, o módulo não fornece informações sobre quais opções são suportadas. Estas opções se referem ao uso de intervalo padrão ou estendido, *Additional Lenght Byte* e Cabeçalho com ou sem informação de endereço;

Em caso de Inicialização *5-Baud* a ferramenta de diagnóstico deve saber quais as opções que estão implementadas. Em caso de Inicialização *Fast* o uso do Cabeçalho e do *Additional Lenght Byte* será igual aos casos de resposta positiva do módulo eletrônico relacionada ao *"StartCommunicationSession"*.

Tabela 8 - Valores possíveis para os *Key Bytes*

3.3.2.4.2 Inicialização com Palavra de Endereço *5-Baud*

3.3.2.4.2.1 Inicialização CARB

Para o uso de CARB, apenas a inicialização *5-Baud* é possível. Esta é uma inicialização funcional onde mensagens são enviadas para todos os módulos relacionados. Para maiores detalhes, consultar ISO 9141-2 e ISO 14230-4.

3.3.2.4.2.2 Inicialização *5-Baud*

3.3.2.4.2.2.1 Geral

A figura a seguir mostra a forma geral da inicialização *5-Baud*. O *byte* de endereço *5-Baud* é transferido da ferramenta de diagnóstico pelas linhas K e L. Após enviar o *byte* de endereço *5-Baud*, a ferramenta de diagnóstico irá manter a linha L em nível alto.

Após receber o *byte* de endereço *5-Baud*, o módulo eletrônico enviará o modelo de sincronização 55h e os dois *Key Bytes* na taxa de comunicação atual. A ferramenta de diagnóstico envia o *Key Bytes* 2 com os *bits* invertidos e o módulo envia o *byte* de endereço com os *bits* invertidos.

No caso de uma inicialização física, o módulo deve responder como mostra a Figura 10.

Para o caso de inicialização funcional, onde mais de um módulo é inicializado, o fabricante deve cuidar para que todos os módulos usem a mesma opção de protocolo. Apenas um módulo pode executar a seqüência de inicialização.

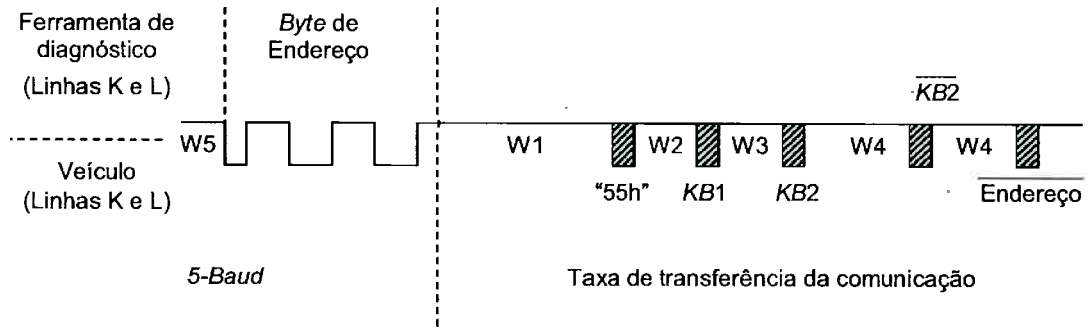


Figura 10 - Inicialização *5-Baud*

A tabela a seguir apresenta os valores e significados dos intervalos da inicialização *5-Baud*. Estes valores são fixos e não podem ser alterados pelo serviço *"AccessCommunicationParameter"*.

Parâmetros de Intervalo	Valores (ms)		Descrição
	Min.	Max.	
W1	60	300	Tempo entre o final do *byte* de endereço e o inicio do modelo de sincronização
W2	5	20	Tempo entre o final do modelo de sincronização e o inicio do *Key Byte* 1.
W3	0	20	Tempo entre os *Key Bytes* 1 e 2
W4	25	50	Tempo entre o *Key Byte* 2 enviado pelo módulo e o mesmo *byte* com os *bits* invertidos enviado pela ferramenta de diagnóstico. Este é o mesmo tempo entre o *Key Byte* 2 invertido pela ferramenta e o endereço com os *bits* invertidos pelo módulo.
W5	300	-	Tempo mínimo anterior ao envio do *byte* de endereço por parte da ferramenta de diagnóstico.

Tabela 9 - Intervalos para Inicialização *5-Baud*

3.3.2.4.2.2.2 *Key Bytes*

São permitidas taxas de transferências entre 1200 e 10400 kbps para comunicação. A ferramenta de diagnóstico irá reconhecer a taxa a partir do *byte* de sincronização (55h).

3.3.2.4.2.2.3 Inicialização Funcional

O objetivo da inicialização funcional é inicializar um grupo de módulos. Os *bytes* de endereço que definem um grupo funcional de módulos seguem as seguintes regras:

- Endereços com valores inferiores a 80h são reservados para padronizações futuras;
- Endereços com valores iguais ou superiores a 80h são especificados pela montadora.

Outros endereços funcionais definidos pela montadora em concordância com a ISO 9141 (paridade ímpar) também são possíveis.

Por razões óbvias, o endereçamento funcional somente é possível quando todos os módulos trabalham na mesma taxa de transferência.

3.3.2.4.2.2.4 Inicialização Física

Este procedimento é válido apenas quando a ferramenta de diagnóstico estabelece comunicação com um único módulo. O endereçamento para inicialização *5-Baud* é especificado na ISO 9141. A paridade ímpar também é possível e os *bytes* de endereço são definidos pela montadora.

3.3.2.4.2.3 Inicialização *Fast*

3.3.2.4.2.3.1 Geral

No caso da Inicialização *Fast*, todos os módulos que serão inicializados devem usar taxa de transferência de 10400 kbps (tanto para inicialização como para comunicação).

A ferramenta de diagnóstico envia um modelo de *Wake-up* (WuP) nas linhas K e L de maneira sincronizada. O modelo inicia após um tempo de repouso na linha K com um curto tempo de Tinil. Após o intervalo Twup, a ferramenta de diagnóstico envia o primeiro *bit* do serviço *"StartCommunication"*, como mostra a figura a seguir:

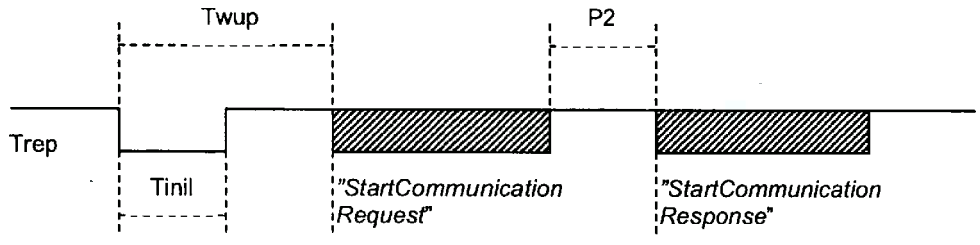


Figura 11 - Inicialização *Fast*

Existem três possibilidades para o intervalo Trep.:

- Primeira transmissão após acionamento da ferramenta de diagnóstico: Trep ≥ W5min;

- Após finalização do serviço *"StopCommunication"*: Trep ≥ P3min;

- Após finalizar a comunicação por estouro (*timeout*) de P3max: Trep ≥ 0.

A tabela a seguir traz os valores de Tinil e Twup:

Parâmetro		Valor Min. (ms)	Valor Max. (ms)
Tinil	25 ± 1 ms	24	26
Twup	50 ± 1 ms	49	51

Tabela 10 - Intervalos para Inicialização *Fast*

Após o envio de um modelo de *Wake-up*, a ferramenta de diagnóstico envia um *"StartCommunicationRequest"* e aguarda a resposta do módulo. A primeira mensagem da Inicialização *Fast* sempre usa o Cabeçalho com endereços de Objetivo e Origem sem o *Additional Lenght Byte*. O módulo que recebe a solicitação pode responder com ou sem informação de endereço e tamanho, informando estes através da indicação de modo suportado nos *Key Bytes*.

3.3.2.4.2.3.2 *Bytes* para Mensagens

As tabelas a seguir descrevem as diferentes mensagens de serviços para o *"StartComunication"*:

Nº do *Byte*	Nome do Parâmetro	CVT[1)]	Vlr Hex.	Mnemonic
1	*Format Byte* Endereçamento físico Endereçamento funcional	M	 $81 $C1	FMT
2	*Target Byte*	M	$XX	TGT
3	*Source Byte*	M	$XX	SRC
4	*"StartCommunication Request Service Id"*	M	$81	SCR
5	*Checksum*	M	$XX	CS
1) Veja 4.3.1				

Tabela 11 - Mensagem *"StartCommunicationRequest"*

Nº do *Byte*	Nome do Parâmetro	CVT[1]	Vlr Hex.	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Byte*	C[2]	$XX	TGT
3	*Source Byte*	C[2]	$XX	SRC
4	*Additional Lenght Byte*	C[3]	$XX	LEN
5	*"StartCommunication Positive Response Service Id"*	S	$C1	SCRPR
6	*Key Byte 1*[4]	M	$XX	KB1
7	*Key Byte 2*[4]	M	$XX	KB2
8	*Checksum*	M	$XX	CS
1) Veja 4.3.1 2) O *Format Byte* é 10XX XXXX ou 11XX XXXX 3) O *Format Byte* é XX00 0000 4) Veja 4.3.2.4.1 referente ao uso dos *Key Bytes*				

Tabela 12 - Mensagem *"StartCommunication Positive Response"*

3.3.3 Serviço *"StopCommunication"*

3.3.3.1 Definição do Serviço

A função deste serviço é finalizar a comunicação de diagnóstico.

3.3.3.2 Tabela de Serviço

Parâmetro	Classificação
"StopCommunication Request"	M
"StopCommunication Positive Response"	S
"StopCommunication Negative Response"	S
Código de Resposta	M

Tabela 13 - Serviço *"StopCommunication"*

3.3.3.3 Procedimento de Serviço

Quando recebe uma indicação inicial de *"StopCommunication"*, o módulo deve verificar se as condições do momento permitem finalizar a comunicação. Neste caso, este deve executar todas as ações necessárias para finalizar a comunicação.

Sendo possível finalizar a comunicação, o módulo deve enviar uma resposta positiva com o parâmetro *"Positive Response"* selecionado antes que a comunicação seja terminada.

Caso a comunicação não possa ser finalizada, o módulo deve enviar uma resposta negativa com o parâmetro *"Negative Response"* selecionado.

3.3.3.4 Implementação

A tabela a seguir apresenta as diferentes mensagens de solicitação para o *"StopCommunication"*.

Nº do *Byte*	Nome do Parâmetro	CVT[1]	Vlr Hex.	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Byte*	C[2]	$XX	TGT
3	*Source Byte*	C[2]	$XX	SRC
4	*Additional Lenght Byte*	C[3]	$XX	LEN
5	*"StopCommunication Request Service Id"*	M	$82	SPR
6	*Checksum*	M	$XX	CS
1) Veja 4.3.1 2) O *Format Byte* é 10XX XXXX ou 11XX XXXX 3) O *Format Byte* é XX00 0000				

Tabela 14 - Mensagem *"StopCommunication Request"*

Nº do *Byte*	Nome do Parâmetro	CVT[1]	Vlr Hex.	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Byte*	C[2]	$XX	TGT
3	*Source Byte*	C[2]	$XX	SRC
4	*Additional Lenght Byte*	C[3]	$XX	LEN
5	*"StopCommunication Positive Response Service Id"*	M	$C2	SPRPR
6	*Checksum*	M	$XX	CS
1) Veja 4.3.1 2) O *Format Byte* é 10XX XXXX ou 11XX XXXX 3) O *Format Byte* é XX00 0000				

Tabela 15 - Mensagem *"StopCommunication Positive Response"*

Nº do *Byte*	Nome do Parâmetro	CVT[1]	Vlr Hex.	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Byte*	C[2]	$XX	TGT
3	*Source Byte*	C[2]	$XX	SRC
4	*Additional Lenght Byte*	C[3]	$XX	LEN
5	*"StopCommunication Negative Response Service Id"*	S	$C1	SPRNR
6	*"StopCommunication Request Service Id"*	M	$82	SPR
7	Código de Resposta[4] = *"generalReject"*	M	$XX=$10	RC
8	*Checksum*	M	$XX	CS
1) Veja 4.3.1 2) O *Format Byte* é 10XX XXXX ou 11XX XXXX 3) O *Format Byte* é XX00 0000 4) Outros códigos de resposta são possíveis: consulte ISO 14230-3				

Tabela 16 - Mensagem *"StopCommunication Negative Response"*

3.3.4 Serviço *"AccessTimingParameter"*

3.3.4.1 Definição do Serviço

O propósito deste serviço é ler e alterar os parâmetros padrões relacionados aos intervalos de tempo da rotina de comunicação quando esta estiver ativa. É considerado um procedimento complexo, que depende da capacidade e da topologia física dos módulos eletrônicos envolvidos.

3.3.4.2 Tabela de Serviço

AccessTimingParameter Request	
Timing Parameter Identifier (TPI)	M
P2min	C[1]
P2max	C[1]
P3min	C[1]
P3max	C[1]
P4min	C[1]
AccessTimingParameter Positive Response	
Timing Parameter Identifier (TPI)	M
P2min	C[2]
P2max	C[2]
P3min	C[2]
P3max	C[2]
P4min	C[2]
AccessTimingParameter Negative Response	S
Response Code	M
Timing Parameter Identifier	M
1) A condição é TPI=Set values	
2) A condição é TPI=Read limits, read current values	

Tabela 17 - Serviço *"AccessTimingParameter"*

3.3.4.3 Procedimento de Serviço

Este procedimento tem quarto modos diferentes:

- Ler limites de parâmetros possíveis de intervalo de tempo;

- Configurar os parâmetros de intervalo de tempo como padrão;

- Ler os parâmetros de intervalo de tempo que estiverem ativos;

- Configurar os parâmetros de intervalo de tempo conforme valores dados.

Quando o módulo receber uma indicação de *"AccessTimingParameter"* com TPI=0, este deve ler os limites do parâmetro de intervalo de tempo que ele deve suportar.

Se o acesso de leitura aos parâmetros de intervalo de tempo for bem sucedido, o módulo deve enviar um *"AccessTimingParameter"* com os parâmetros *Positive Response*.

Do contrário, se o acesso de leitura aos parâmetros de intervalo de tempo for mal sucedido, o módulo deve enviar um *"AccessTimingParameter"* com os parâmetros *Negative Response*.

Quando receber uma indicação de *"AccessTimingParameter"* com TPI=1, o módulo deve alterar todos os parâmetros de intervalo de tempo para valores padrões e enviar um *"AccesTimingParameter"* com os parâmetros de *Positive Response* antes que os parâmetros carregados com valores padrões se tornem ativos.

Se os parâmetros não puderem ser alterados para os valores padrões por qualquer razão, o módulo deve manter a comunicação e enviar um *"AccessTimingParameter"* com os parâmetros de *Negative Response*.

Após receber um *"AccessTimingParameter"* com TPI=2, o módulo deve ler os valores atuais que estão sendo usados nos parâmetros de intervalo de tempo.

Se o acesso de leitura aos parâmetros de intervalo de tempo for bem sucedido, o módulo deve enviar um *"AccessTimingParameter"* com os parâmetros *Positive Response*.

Mais uma vez, por questões óbvias, se o acesso de leitura aos parâmetros de intervalo de tempo for mal sucedido, o módulo deve enviar um *"AccessTimingParameter"* com os parâmetros *Negative Response*.

Uma vez recebido um *"AccessTimingParameter"* com TPI=3, o módulo deve verificar se os parâmetros de intervalo de tempo podem ser alterados nas condições presentes neste momento.

Se as condições forem válidas, o módulo deve executar todas as ações necessárias para alterar os parâmetros e enviar um *"AccessTimingParameter"* com os parâmetros *Positive Response* antes que os parâmetros com os novos valores se tornem ativos.

Como anteriormente, se os parâmetros não puderem ser alterados para os valores desejados por qualquer razão, o módulo deve manter a comunicação e enviar um *"AccessTimingParameter"* com os parâmetros *Negative Response*.

3.3.4.4 Implementação

A tabela a seguir apresenta a seleção dos modos *read, write, current* e *limits* através do *Timing Parameter Identifier*:

Modo	TPI	CVT[1)]
Ler os limites	0000 0000	C[2)]
Configurar os parâmetros para valores padrões	0000 0001	
Ler os valores atuais	0000 0010	C[3)]
Configurar valores	0000 0011	C[2)]
1) Veja 5.1		
2) Os parâmetros de intervalo de tempo são incluídos na mensagem se TPI=3		
3) Os parâmetros de intervalo de tempo são incluídos na mensagem se TPI=0 ou 2		

Tabela 18 - Modo de Seleção

3.3.4.5 *Bytes* da Mensagem

As tabelas de 19 a 21 descrevem as diferentes mensagens *"AccessTimingParameter"*.

Byte	Nome do Parâmetro	CVT[1]	Vlrs Hex	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Address Byte*	C[2]	$XX	TGT
3	*Source Address Byte*	C[2]	$XX	SRC
4	*Additional Length Byte*	C[3]	$XX	LEN
5	*"AccessTimingParameter" Request Service Id*	S	$83	ATP
6	*Timing Parameters Identifier*	M	$0X[5]	TPI
7	P2min	C[4]	$XX	P2min
8	P2max	C[4]	$XX	P2max
9	P3min	C[4]	$XX	P3min
10	P3max	C[4]	$XX	P3max
11	P4min	C[4]	$XX	P4min
12	*Checksum*	M	$XX	CS
1) Veja 5.1				
2) *Format Byte* é 10XX XXXX ou 11XX XXXX				
3) *Format Byte* é XX00 0000				
4) Os parâmetros de intervalo de tempo são incluídos na mensagem se TPI=3				
5) Onde X pode ser 0 (ler limites), 1 (configurar parâmetros com valores padrões), 3 (ler parâmetros ativos), 4 (configurar parâmetros com valores desejados)				

Tabela 19 - Mensagem *"AccessTimingParameter" Request*

Byte	Nome do Parâmetro	CVT[1]	Vlrs Hex	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Address Byte*	C[2]	$XX	TGT
3	*Source Address Byte*	C[2]	$XX	SRC
4	*Additional Length Byte*	C[3]	$XX	LEN
5	*"AccessTimingParameter" Positive Response Service Id*	M	$C3	ATPPR
6	*Timing Parameters Identifier*	M	$0X[5]	TPI
7	P2min	C[4]	$XX	P2min
8	P2max	C[4]	$XX	P2max
9	P3min	C[4]	$XX	P3min
10	P3max	C[4]	$XX	P3max
11	P4min	C[4]	$XX	P4min
12	*Checksum*	M	$XX	CS
1) Veja 5.1				
2) *Format Byte* é 10XX XXXX ou 11XX XXXX				
3) *Format Byte* é XX00 0000				
4) Os parâmetros de intervalo de tempo são incluídos na mensagem se TPI=0 ou 2				
5) Onde X pode ser 0 (ler limites), 1 (configurar parâmetros com valores padrões), 3 (ler parâmetros ativos), 4 (configurar parâmetros com valores desejados)				

Tabela 20 - Mensagem *"AccessTimingParameter" Positive Response*

Byte	Nome do Parâmetro	CVT[1]	Vlrs Hex	Mnemonic
1	*Format Byte*	M	$XX	FMT
2	*Target Address Byte*	C[2]	$XX	TGT
3	*Source Address Byte*	C[2]	$XX	SRC
4	*Additional Length Byte*	C[3]	$XX	LEN
5	*Negative Response Service Id*	S	$7F	ATPNR
6	*"AccessTimingParameter" Request Service Id*	M	$10	ATP
7	*Response Code*[4] = *generalReject*	C[4]	$10	RC
8	*Checksum*	M	$XX	CS
1) Veja 5.1				
2) *Format Byte* é 10XX XXXX ou 11XX XXXX				
3) *Format Byte* é XX00 0000				
4) Outros *Response Codes* são possíveis. Consulte ISO 14230-3				
5) Onde X pode ser 0 (ler limites), 1 (configurar parâmetros com valores padrões), 3 (ler parâmetros ativos), 4 (configurar parâmetros com valores desejados)				

Tabela 21 - Mensagem *"AccessTimingParameter" Negative Response*

3.3.5 Serviço *"SendData"*

3.3.5.1 Definição

O propósito deste serviço é transmitir o dado a partir de um serviço de solicitação e através de uma conexão para comunicação baseada no protocolo "KW2000".

3.3.5.2 Tabela de Serviço

A tabela a seguir ilustra os serviços relacionados ao *"SendData"*:

SendData Request	M
Service Data	M
SendData Positive Response	S
SendData Negative Response	S
Response Code	M

Tabela 22 - Serviço *"SendData"*

3.3.5.3 Procedimento do Serviço

Após uma solicitação *"SendData"* de uma aplicação de camada, a respectiva camada de conexão de dados do transmissor da mensagem irá executar todos os passos necessários para transmitir os parâmetros da solicitação através de uma mensagem KWP2000. Isto inclui a definição do cabeçalho (com o *Format Byte*), os dados da mensagem e o cálculo do *Checksum*, identificação de modo *Idle*, a transmissão dos *bytes* da mensagem e a determinação do intervalo de tempo.

Uma vez recebido a mensagem pela conexão definida pelo protocolo "KW2000", a respectiva camada de conexão de dados do receptor da mensagem irá executar todas as ações necessárias para prover à camada de aplicação a informação recebida. Isto inclui o reconhecimento do início da mensagem, os intervalos de tempo, a recepção dos *bytes* da mensagem, a verificação do *checksum*, segmentação dos dados baseado nas informações do *Format Byte* e entrega dos dados da mensagem à camada de aplicação com a indicação *"SendData"*.

Se o serviço foi completado com sucesso, ou seja, a mensagem foi transmitida, um *"SendData"* com o parâmetro *Positive Response* selecionado é entregue pela camada de conexão do dispositivo de transmissão à respectiva camada de aplicação.

Porém, se o serviço não pode ser executado pela camada de conexão do dispositivo de transmissão, um *"SendData"* com o parâmetro *Negative Response* selecionado é entregue à respectiva camada de aplicação.

3.3.6 Manuseando Erros

3.3.6.1 Serviço *"StartCommunication"*

Se a ferramenta de diagnóstico identificar um erro no Serviço *"StartCommunication"* seja proveniente do intervalo de tempo, seja por erro nos dados, esta deve aguardar por um período de W5 antes de iniciar o processo novamente (através do modelo de *Wake-up*). Se

um módulo detectar um erro proveniente da ferramenta de diagnóstico, este deve ser imediatamente preparado para reconhecer um outro Serviço *"StartCommunication"*.

Tanto a ferramenta de diagnóstico como o módulo devem estar aptos a reconhecer falhas e aceitar valores máximos de intervalos de tempo. Valores mínimos de erro de intervalos de tempo podem ser descartados, mas poderão causar erros na identificação dos *bits*.

3.3.6.2 Práticas comuns na comunicação

É permitido, mas não mandatório que a ferramenta de diagnóstico e os módulos possam monitorar suas próprias mensagens. Isto é normalmente utilizado onde o manuseio de erro na fase de comunicação deve ser definido.

3.3.6.3 Módulo detecta erros provenientes da ferramenta de diagnóstico.

O módulo deve verificar cada mensagem recebida através do *Checksum* e número de *bytes* recebidos antes que o intervalo P2max seja executado. Se ambos estiverem errados, então o módulo não deve enviar resposta alguma e deve ignorar a mensagem completamente. Não é mandatório que o módulo verifique outros intervalos de tempo, mas isto pode ser feito se conveniente. Caso um erro seja identificado neste caso, mais uma vez nenhuma mensagem deve ser enviada.

O módulo pode detectar outros erros como de formato ou conteúdo das mensagens, e estes devem satisfazer as condições de *Checksum* e tamanho. Neste caso, para que a ferramenta de diagnóstico possa ser informada que não há apenas um simples problema de comunicação, o módulo deve responder com a resposta negativa apropriada.

3.3.6.4 Ferramenta de diagnóstico detecta erro na resposta do veículo

Uma solicitação pode resultar em uma simples resposta de um simples módulo ou em múltiplas respostas de múltiplos módulos. A ferramenta de diagnóstico deve certificar que todas as respostas são corretas em termos de tamanho e *Checksum*. Em caso de erro ou falta de resposta durante P2max, ela deve retransmitir a mensagem original duas vezes (totalizando três transmissões) antes de considerar que há um erro mais severo ocorrendo. A camada de aplicação deve ser informada dos erros na conexão de comunicação, o que permitirá à ela executar os passos apropriados para cada caso.

3.3.6.5 Módulo detecta erro na resposta do módulo

O módulo pode detectar diferenças entre o que ele transmitiu e o que foi detectado na linha K. Neste caso ele não deve agir ou deve simplesmente tentar retransmitir a informação dentro de um período P2 após cessar as atividades no barramento. Isto permite a adaptação de vários métodos de gerenciamento de barramento.

3.3.6.6 Ferramenta de diagnóstico detecta erros de sua própria transmissão

A ferramenta de diagnóstico pode detectar diferenças entre o que ela transmitiu e o que foi detectado na linha K. Neste caso ela deve retransmitir a mensagem inteira e indicar um erro à camada de aplicação. Após um tempo de P2max, a ferramenta deve retransmitir a solicitação.

4 Aplicação do Protocolo "KW2000"

Como desfecho do estudo do protocolo "KW2000", foi desenvolvido um programa em "Delphi" para leitura dos dados do módulo de controle do motor de um veículo popular que trabalha com o protocolo "KW2000" como padrão para troca de dados de diagnóstico.

4.1 Introdução

Através da observação da ferramenta de diagnóstico utilizada em concessionárias, foi possível notar que, basicamente, o módulo de controle do motor do veículo estudado possuía as seguintes funções:

- Leitura e remoção de códigos de falha;
- Exibição de dados;
- Tomada de Transiente;
- Teste de atuadores;
- Identificação do módulo eletrônico;
- Exibição do estado do imobilizador;
- Verificação de sobre-giro do motor;
- Programação do VIN;
- Programação do tipo de combustível.

Para verificar o funcionamento do protocolo no tocante à troca de informações de diagnóstico, as seguintes funções foram selecionadas para implementação do software em "Delphi":

- Exibição de dados;
- Identificação do módulo eletrônico;

A estratégia para seleção destas duas funções baseou-se no número de parâmetros envolvidos em cada uma e a possibilidade de visualização imediata da alteração dos parâmetros colhidos. No total são mais de 100 *bytes* de dados, sendo que 55 parâmetros que são parte destes *bytes* necessitam ser atualizados instantaneamente e mostrados na tela do computador, propiciando a análise do comportamento do veículo em funcionamento.

A seguir serão apresentados os passos executados para a conclusão deste software.

4.2 Objetivo

O objetivo desta aplicação foi examinar na prática o comportamento de dispositivos eletrônicos trocando informações através do protocolo "KW2000".

4.3 Desenvolvimento

O desenvolvimento desta aplicação foi realizado sobre a seguinte estrutura física:

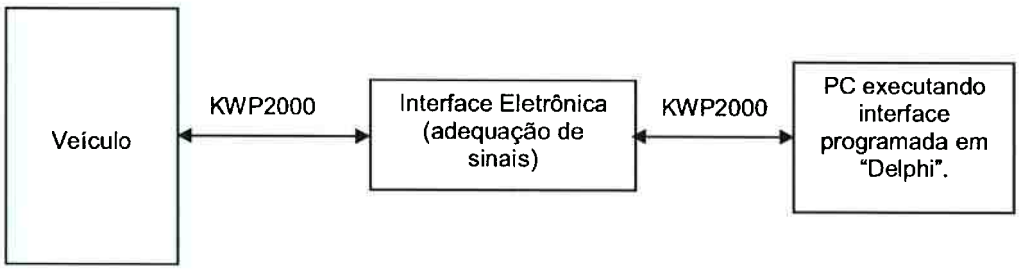


Figura 12 - Estrutura da aplicação desenvolvida

O veículo possui vários módulos eletrônicos de controle. Neste trabalho o alvo é o módulo de controle do motor localizado no compartimento do motor do veículo, cuja linha de diagnóstico está posicionada juntamente com as linhas dos demais módulos do veículo no conector de diagnóstico que pode ser encontrado ao lado da caixa principal de fusíveis, conforme apresentado na figura abaixo.

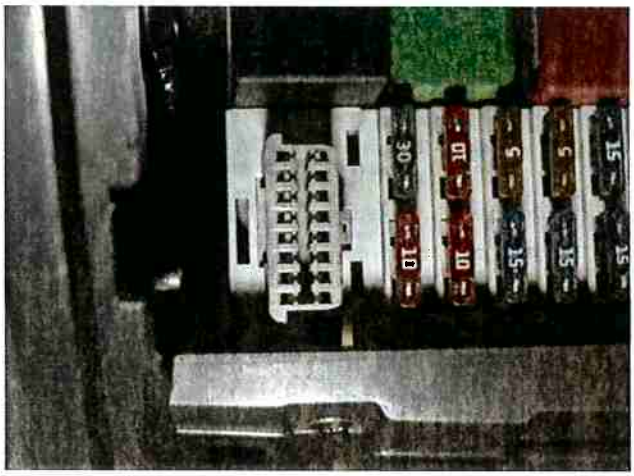


Figura 13 - Conector de diagnóstico

Com esta estrutura definida, foram realizados testes de navegação com a ferramenta de diagnóstico dedicada, utilizada por toda a rede de concessionárias de uma das maiores montadoras do país. Estes testes tiveram por finalidade a verificação do funcionamento da ferramenta existente, bem como a visualização dos parâmetros relacionados ao motor do veículo.

A partir destes testes, foi desenvolvida a interface em "Delphi", conforme segue:

4.3.1 Taxa de transferência e níveis dos sinais elétricos

Como um dos tópicos relacionados na introdução deste trabalho, estudou-se a compatibilidade dos níveis de sinais da porta serial do computador com os níveis de sinais do módulo de controle do motor, bem como a taxa de transferência exigida pelo módulo (10400 bps – enquadrando-se no padrão ditado pelo protocolo quando se utiliza inicialização *Fast*), que é uma taxa fora dos padrões. Inicialmente cogitou-se a implementação de uma interface eletrônica com microcontrolador para adequar os sinais e atingir a taxa de transferência exigida. Porém, em consulta a profissionais que trabalham diretamente com programas de desenvolvimento de calibrações de módulos de controle de motores, foi

possível notar a utilização de uma interface simples, confeccionada com transistores para adequação dos níveis de sinais, a qual por motivos éticos não será divulgada neste trabalho, uma vez que o objetivo deste é a verificação do comportamento do protocolo "KW2000" e não o desenrolar de problemas relacionados com níveis de sinais elétricos.

Uma vez descoberta a interface, a taxa de transferência foi alterada para 10400 bps através das configurações do "Delphi", sendo possível notar que o módulo de controle do motor respondia positivamente aos comandos enviados.

4.3.2 Intervalos de tempo

Relembrando o embasamento teórico deste trabalho, para a inicialização *Fast* (que foi a escolhida para esta aplicação) o ciclo de *Wake-up* deve respeitar o seguinte formato:

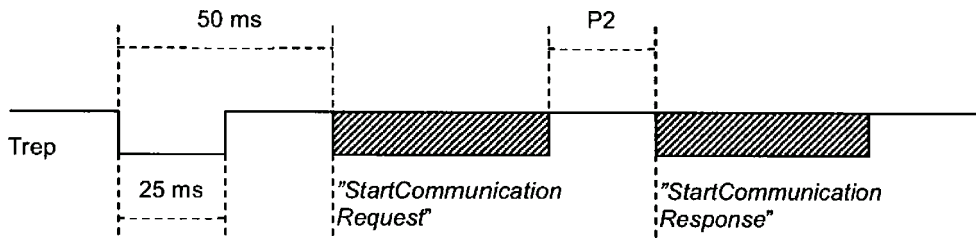


Figura 14 - Ciclo *Wake-up*

Embora o "Delphi" apresente temporizadores com base de contagem de 1 ms, notou-se, através da utilização de um osciloscópio, que tais temporizadores na verdade possuem suas bases de tempo limitadas a 10 ms. Isto caracterizou um grande problema, uma vez que o ciclo de *Wake-Up*, exigido pelo protocolo, força dois intervalos de 25 ms antes do envio de qualquer mensagem de solicitação de serviço. Como solução, recorreu-se aos parâmetros do Microsoft Windows para contagem do tempo exato.

4.3.3 Desenvolvimento do Software

Seguindo a idéia de implementação das funções de "Exibição de dados" e "Identificação do módulo eletrônico", levantou-se quais os serviços que deveriam ser executados para solicitação destas informações ao módulo de controle do motor. São eles:

- *"StartCommunication"*;
- *"ReadEcuIdentification"*;
- *"ReadDataByLocalIdentifier"*

Através do software de Engenharia "Carta", foi possível ler os dados trocados entre o *notebook* e o módulo de controle do motor do veículo avaliado. Para o serviço *"StartCommunication"*, tivemos a seguinte leitura:

<REQUEST> STC – *"StartCommunication"* :

1, 49.576, <81> --> *With address info, physical addressing*
2, 5.277, <11> --> *target address*: 11h
3, 5.181, <F1> --> *source address*: F1h
4, 5.181, <81> --> sid: 81h
5, 5.181, <04> --> *calculated checksum*: 04h

Os valores acima representam os 5 *bytes* enviados através da porta serial pelo software desenvolvido. O significado de cada valor apresentado pode ser explicado da seguinte forma:

Primeira Coluna: Seqüência de acontecimentos. O número 1 representa o primeiro *byte* enviado, o número dois o segundo e assim por diante;

Segunda Coluna: Intervalo de tempo (em ms) entre os *bytes*. O primeiro valor (49,576 ms) significa que o primeiro *byte* (*Format Byte*) foi enviado 49,576 ms após a primeira atuação da porta serial, o que respeita o ciclo total de *Wake-up* que é padronizado como 50 ms (± 1).

O segundo valor (5,277 ms) mostra que o *Target Byte* foi enviado 5,277 ms após o *Format Byte*, o que comprova a observação do intervalo de tempo P4 entre os *bytes* da mesma mensagem, que é padronizado no intervalo de 5 a 20 ms. Da mesma forma temos os *bytes* 3, 4 e 5 respeitando os intervalos exigidos pelo protocolo "KW2000".

Terceira Coluna: Valor em hexadecimal do *byte* enviado pelo *notebook* para o módulo. O primeiro *byte* (*Format Byte*) possui o valor 81h, que em binário representa 10000001, cujo significado pode ser detalhado recorrendo-se ao conceito do *Format Byte* (Tabela 1), ou seja:

Format Byte:

A1	A0	L5	L4	L3	L2	L1	L0
7	6	5	4	3	2	1	0

Substituindo:

1	0	0	0	0	0	0	1
7	6	5	4	3	2	1	0

Portanto:

A1 e A0 valem respectivamente 1 e 0. Recorrendo-se à Tabela 1, isto caracteriza "Com informação física de endereçamento", ou seja, os *bytes* de origem e destino contém endereçamentos físicos.

L5, L4, L3, L2, L1 e L0 valem respectivamente 0, 0, 0, 0, 0 e 1. Na Tabela 2, vemos que quando estes *bits* não estão zerados, eles representam a quantidade de *bytes* de dados (incluindo o SId), não sendo necessário utilizar o *byte* de tamanho adicional (*Additional Lenght Byte*). Assim, significa que existe apenas um *byte* de dados: o SId.

Em seguida há a presença do *Target Byte*, que representa o endereço físico do módulo de controle do motor (11h). Após ele o *Source Byte*, apresentando o endereço físico da ferramenta de diagnóstico (no caso o *notebook* – F1h). Em seguida encontra-se o *Service Identification* (SId) que vale 81h, valor este correspondente ao serviço *"StartCommunication"*

(vide Tabela 10). E por fim o *Checksum* que representa a soma simples de todos os *bytes* enviados (excluindo o próprio *Checksum*) nesta mensagem, limitado a 2 algarismos em hexadecimal:

81h + 11h + F1h + 81h = **204**h, logo o "Checksum" vale 04h.

Como resposta do módulo, colheu-se os seguintes dados:

<RESPONSE> STCPR – *"StartCommunication" Positive Response* :

6, 32.935, <83> --> *With address info, physical addressing*
7, 0.093, <F1> --> *target address*: F1h
8, 0.094, <11> --> *source address*: 11h
9, 0.093, <C1> --> sid: C1h
10, 0.093, <EF> --> *Keybyte* 1: EFh
11, 0.093, <8F> --> *Keybyte* 2: 8Fh
12, 0.093, <C4> --> *calculated checksum*: C4h

Mais uma vez foi possível notar o cumprimento dos intervalos de tempo exigidos pelo protocolo "KW2000". O primeiro valor (32,935 ms) representa P2 (limitado entre 25 e 50 ms – vide Tabela 3). Os demais (por volta de 0,093 ms) representam P1 (limitado entre 0 e 20 ms – vide Tabela 3).

Quanto aos valores:
- o *byte* de formato recebido foi 83h, que significa endereçamento físico com 3 *bytes* de dados;
- Objetivo igual a F1h, que é o endereço do *notebook*;
- Fonte igual a 11h, que representa o endereço do módulo que está respondendo;
- *Service Identifier* igual a C1h, que caracteriza resposta positiva à solicitação de início de comunicação (*"StartCommunication"* – vide Tabela 11);
- *Key Byte* 1 igual a EFh e *Key Byte* 2 igual a 8Fh, que representa, segundo a Tabela 7:
    - Tamanho da Informação suportado: *Format Byte* e *Additional Lenght Byte* (Ambos os modos);
    - Tipo de cabeçalho suportado: Com e sem informação de endereço (Ambos os tipos);
    - Intervalo: Normal.
- *Checksum*: 83h + F1h + 11h + C1h + EFh + 8Fh = 3**C4**h, portanto igual a C4h.

Estabelecida a comunicação, enviou-se o serviço desejado para leitura dos dados do motor do veículo (*"ReadDataByLocalIdentifier"*) e da identificação do módulo de controle do motor (*"ReadEcuIdentification"*).

Para executar a leitura dos dados do motor, o software desenvolvido enviou a seguinte seqüência de *bytes*:

<REQUEST> RDBLI – *"ReadDataByLocalIdentifier"* :

13, 56.577, <80> --> *With address info, physical addressing*
14, 5.181, <11> --> *target address*: 11h
15, 5.181, <F1> --> *source address*: F1h
16, 5.180, <02> --> *length byte*: 02h
17, 5.181, <21> --> sid: 21h
18, 5.181, <01> ...
19, 5.180, <A6> --> *calculated checksum*: A6h

Neste caso, para verificar a aplicação do *byte* de tamanho adicional (*Additional Lenght Byte*), o *byte* de formato foi enviado com apenas informações sobre o tipo de endereçamento, sendo a quantidade de *bytes* de dados informada pelo *Additional Lenght Byte*.

Os *bytes* significam:

- *Byte* de formato igual a 80h, que indica endereçamento físico. Agora foi a vez do intervalo P3 ser respeitado (variação permitida entre 55 e 5000 ms) conforme representado na figura 6;
- Objetivo igual a 11h, que é o endereço do módulo controlador;
- Fonte igual a F1h, que representa o endereço do *notebook*;
- *Byte* de tamanho adicional igual a 02h, que indica que a mensagem de dados terá 2 *bytes*.
- SId igual a 21h, que indica o serviço solicitado (*"ReadDataByLocalIdentifier"* – conforme ISO14230-3);
- Primeiro e único *byte* de mensagem igual a 01h. Este *byte* é exigido pelo serviço *"ReadDataByLocalIdentifier"* denominado *"RecordLocalIdentifier"* (segundo ISO14230-3). Seu conteúdo não é especificado por norma e sim por cada montadora.
- Soma para *Checksum* igual a 1**A6**h, portanto *Checksum* igual a A6h.

Como resposta do módulo encontrou-se:

<RESPONSE> RDBLIPR - *"ReadDataByLocalIdentifier" Positive Response* :

20, 37.878, <80> --> *With address info, physical addressing*
21, 0.092, <F1> --> *target address*: F1h
22, 0.093, <11> --> *source address*: 11h
23, 0.093, <60> --> *length byte*: 60h
24, 0.093, <61> --> sid: 61h
25, 0.093, <01> ...
26, 0.093, <01> ...
27, 0.093, <16> ...
28, 0.093, <14> ...
29, 0.093, <20>
30, 0.093, <00> ...
31, 0.093, <00> ...
32, 0.093, <00> ...
33, 0.093, <00> ...
34, 0.093, <00> ...
35, 0.093, <00> ...
36, 0.093, <02> ...
37, 0.093, <00> ...
38, 0.093, <00> ...
39, 0.093, <A2> ...
40, 0.093, <00> ...
41, 0.093, <00> ...
42, 0.093, <A1> ...
43, 0.093, <0F> ...
44, 0.093, <00> ...
45, 0.092, <08> ...
46, 0.093, <21> !
47, 0.093, <01> ...
48, 0.093, <00> ...
49, 0.093, <00> ...
50, 0.093, <01> ...
51, 0.093, <01> ...
52, 0.092, <00> ...

53, 0.093, <02> ...
54, 0.093, <00> ...
55, 0.093, <18> ...
56, 0.093, <00> ...
57, 0.093, <74> t
58, 0.093, <00> ...
59, 0.093, <00> ...
60, 0.093, <67> g
61, 0.093, <8E> ...
62, 0.093, <00> ...
63, 0.093, <80> €
64, 0.093, <6A> j
65, 0.093, <41> A
66, 0.093, <9E> ...
67, 0.093, <42> B
68, 0.092, <00> ...
69, 0.093, <00> ...
70, 0.093, <00> ...
71, 0.093, <C3> ...
72, 0.093, <E0> ...
73, 0.093, <5C> \
74, 0.093, <00> ...
75, 0.093, <00> ...
76, 0.093, <00> ...
77, 0.093, <00> ...
78, 0.093, <1E> ...
79, 0.093, <00> ...
70, 0.093, <00> ...
81, 0.093, <00> ...
82, 0.093, <00> ...
83, 0.093, <00> ...
84, 0.093, <00> ...
85, 0.093, <00> ...
86, 0.093, <00> ...
87, 0.093, <00> ...
88, 0.093, <00> ...
89, 0.093, <33> 3

90, 0.093, <5A> Z
91, 0.093, <43> C
92, 0.093, <85> ...
93, 0.092, <33> 3
94, 0.093, <20>
95, 0.093, <CD> ...
96, 0.093, <32> 2
97, 0.093, <00> ...
98, 0.093, <00> ...
99, 0.093, <00> ...
100, 0.093, <00> ...
101, 0.093, <00> ...
102, 0.093, <00> ...
103, 0.093, <00> ...
104, 0.093, <00> ...
105, 0.093, <00> ...
106, 0.093, <51> Q
107, 0.093, <51> Q
108, 0.093, <80> €
109, 0.093, <00> ...
110, 0.093, <64> d
111, 0.092, <00> ...
112, 0.093, <00> ...
113, 0.093, <00> ...
114, 0.093, <69> i
115, 0.093, <04> ...
116, 0.093, <00> ...
117, 0.092, <80> €
118, 0.093, <62> b
119, 0.093, <12> ...
120, 0.093, <48> --> *calculated checksum*: 48h

Os *bytes* representam:

- *Byte* de formato igual a 80h, indicando endereçamento físico e necessidade da utilização do *byte* de tamanho adicional (conforma descrito na Tabela 2);
- Objetivo igual a F1h, que é o endereço estipulado para o *notebook*;

- Fonte igual a 11h, que representa o endereço do módulo que está respondendo;
- *Byte* de tamanho adicional igual a 60h, indicando 96 *bytes* de dados incluindo o SId;
- SId igual a 61h, indicando resposta positiva do módulo frente a solicitação *"ReadDataByLocalIdentifier"* (conforme ISO14230-3);
- *Bytes* de número 25 a 119 representam toda a informação dos dados do motor que estão disponíveis neste serviço. Itens como rotação do motor, temperatura do líquido de arrefecimento, posição da borboleta de aceleração, quantidade de combustível no tanque, etc. estão neste pacote enviado ao *notebook* pelo módulo;
- *Checksum* calculado igual a 48h.

Desta forma, aplicando-se as equações necessárias sobre cada *byte* recebido (conforme especificação técnica do módulo), foi possível mostrar na tela do *notebook* os mesmos dados do motor estudado, lidos através da ferramenta de diagnóstico nas concessionárias e oficinas. Esta tela é apresentada abaixo:

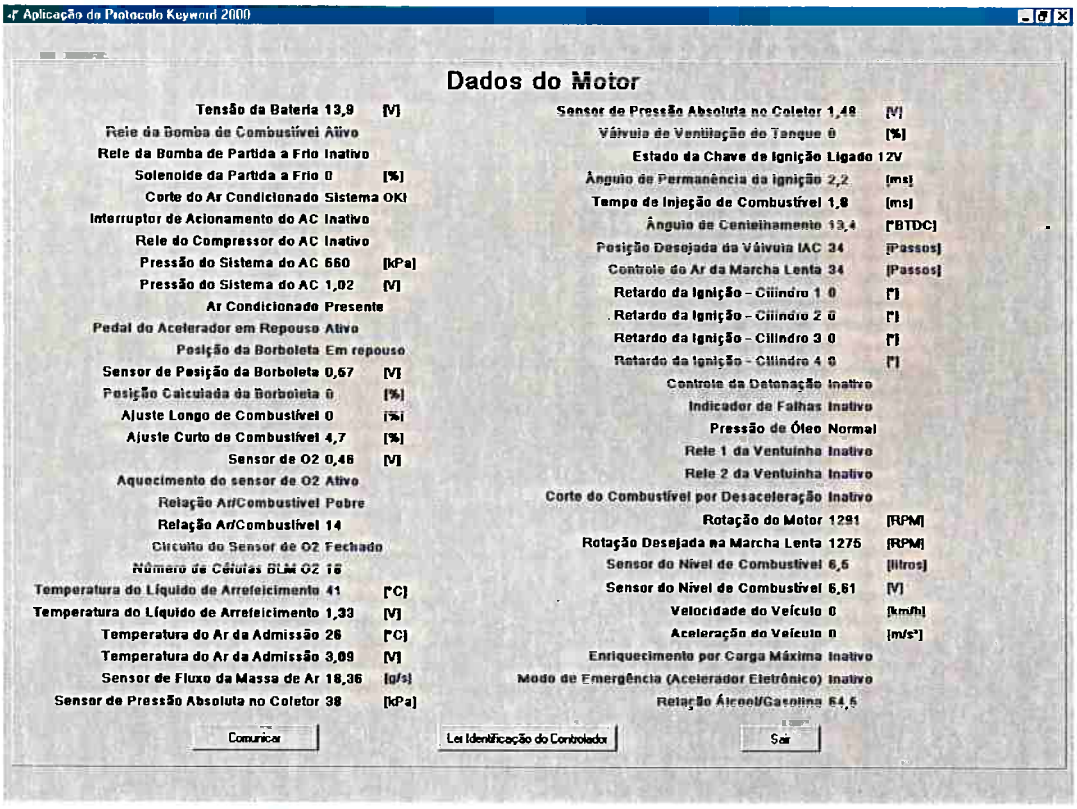


Figura 15 - Tela de leitura dos dados do motor

Com os dados do motor lidos corretamente, aplicou-se o serviço *"ReadEcuIdentification"* para captar os dados referentes à identificação do módulo de controle.

Para solicitar este serviço, os seguintes *bytes* foram enviados através do software desenvolvido:

<REQUEST> REI - *"ReadEcuIdentification"* :

121, 55.879, <80> --> *With address info, physical addressing*
122, 5.182, <11> --> *target address*: 11h
123, 5.277, <F1> --> *source address*: F1h
124, 5.181, <02> --> *length byte*: 02h
125, 5.181, <1A> --> sid: 1Ah
126, 5.181, <80> --> IDOPT = 80h *reportECUIdentificationDataTable*
127, 5.181, <1E> --> *calculated checksum*: 1Eh

Mais uma vez foi utilizado o *byte* de tamanho adicional para indicar a quantidade de *bytes* na mensagem enviada.

A seguir o significado dos *bytes*:

- *Byte* de formato igual a 80h, indicando endereçamento físico e o uso do *byte* de tamanho adicional;
- Objetivo igual a 11h, que é o endereço do módulo controlador;
- Fonte igual a F1h, que representa o endereço do *notebook*;
- *Byte* de Tamanho adicional igual a 02h, indicando a existência de apenas 2 *bytes* na mensagem de dados;
- SId igual a 1Ah, que representa o serviço solicitado (*"ReadEcuIdentification"*) conforme a ISO14230-3;
- Primeiro e único *byte* de mensagem igual a 80h. Este *byte* é parte integrante do processo de solicitação do serviço *"ReadEcuIdentification"* e é chamado de "*IdentificationOption*", sendo seu conteúdo especificado pela montadora;
- *Checksum* calculado igual a 1Eh.

Como resposta do módulo:

<RESPONSE> REIPR - *"ReadEcuIdentification" Positive Response* :

128, 31.856, <80> --> *With address info, physical addressing*
129, 0.093, <F1> --> *target address*: F1h

130, 0.093, <11> --> *source address*: 11h
131, 0.093, <58> --> *length byte*: 58h
132, 0.093, <5A> --> sid: 5Ah
133, 0.093, <80> €
134, 0.093, <38> 8
135, 0.093, <42> B
136, 0.093, <47> G
137, 0.093, <52> R
138, 0.093, <4D> M
139, 0.093, <36> 6
140, 0.093, <39> 9
141, 0.093, <39> 9
142, 0.093, <30> 0
143, 0.093, <37> 7
144, 0.093, <47> G
145, 0.093, <31> 1
146, 0.093, <34> 4
147, 0.093, <31> 1
148, 0.093, <33> 3
149, 0.093, <32> 2
150, 0.093, <38> 8
151, 0.092, <53> S
152, 0.093, <30> 0
153, 0.093, <30> 0
154, 0.093, <31> 1
155, 0.093, <30> 0
156, 0.093, <30> 0
157, 0.093, <39> 9
158, 0.093, <36> 6
159, 0.093, <36> 6
160, 0.093, <34> 4
161, 0.093, <20>
162, 0.093, <06> ...
163, 0.093, <11> ...
164, 0.093, <08> ...
165, 0.093, <44> D
166, 0.093, <45> E

167, 0.093, <4C> L
168, 0.093, <20>
169, 0.093, <20>
170, 0.093, <30> 0
171, 0.093, <31> 1
172, 0.093, <31> 1
173, 0.093, <46> F
174, 0.093, <33> 3
175, 0.093, <31> 1
176, 0.093, <30> 0
177, 0.092, <36> 6
178, 0.093, <33> 3
179, 0.093, <30> 0
180, 0.093, <39> 9
181, 0.093, <34> 4
182, 0.093, <37> 7
183, 0.093, <30> 0
184, 0.093, <32> 2
185, 0.093, <33> 3
186, 0.093, <33> 3
187, 0.093, <31> 1
188, 0.093, <20>
189, 0.093, <5A> Z
190, 0.093, <43> C
191, 0.093, <46> F
192, 0.093, <46> F
193, 0.093, <52> R
194, 0.093, <4B> K
195, 0.093, <00> ...
196, 0.093, <01> ...
197, 0.093, <20>
198, 0.093, <20>
199, 0.093, <20>
200, 0.093, <20>
201, 0.092, <20>
202, 0.093, <20>
203, 0.093, <58> X

204, 0.093, <31> 1
205, 0.093, <30> 0
206, 0.093, <59> Y
207, 0.093, <46> F
208, 0.093, <4C> L
209, 0.093, <00> ...
210, 0.093, <1F> ...
211, 0.093, <64> d
212, 0.093, <50> P
213, 0.093, <42> B
214, 0.093, <52> R
215, 0.093, <32> 2
216, 0.092, <30> 0
217, 0.093, <30> 0
218, 0.093, <37> 7
219, 0.093, <20>
220, 0.093, <4B> --> *calculated checksum*: 4Bh

O significado de cada *byte* é:

- *Byte* de tamanho igual a 80h, indicando endereçamento físico e a presença do *byte* de tamanho adicional;
- Objetivo igual a F1h, que é o endereço do *notebook*;
- Fonte igual a 11h, que representa o endereço do módulo que está respondendo;
- *Byte* de tamanho adicional igual a 58h, indicando que seriam 88 *bytes* de dados incluindo o SId;
- SId igual a 5Ah, caracterizando resposta positiva do módulo frente a solicitação *"ReadEcuIdentification"* (conforme ISO14230-3);
- *Bytes* de número 133 a 219 representam toda a informação referente à identificação do controlador que estão disponíveis neste serviço. Itens como número de identificação do veículo (VIN), número do software, identificador, tipo de motor, etc. estão neste pacote enviado pelo módulo ao *notebook*. Conforme pode ser observado ao lado direito do conteúdo de cada *byte*, os valores representam caracteres em código ASCII;
- *Checksum* calculado igual a 4Bh.

Desta forma, organizando os *bytes* recebidos conforme especificação técnica do módulo, foi possível mostrar na tela do *notebook* as mesmas informações sobre identificação do módulo controlador do motor, lidas através da ferramenta de diagnóstico nas concessionárias. Esta tela é apresentada abaixo:

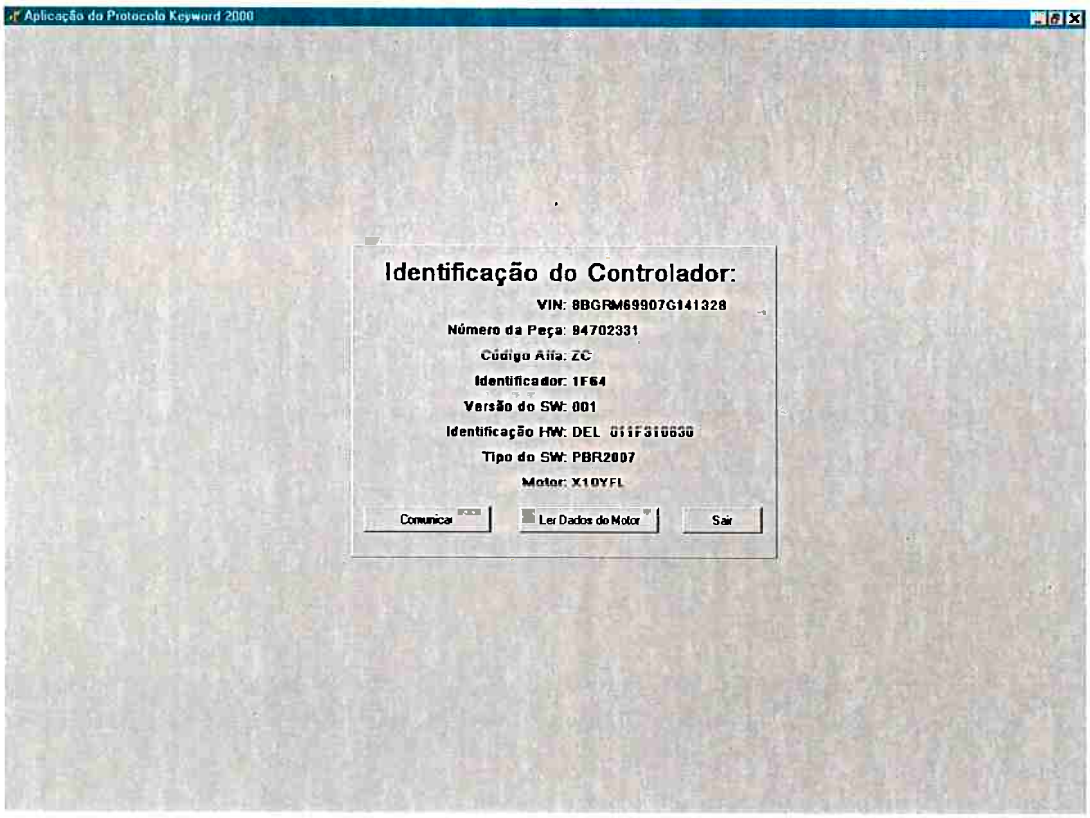
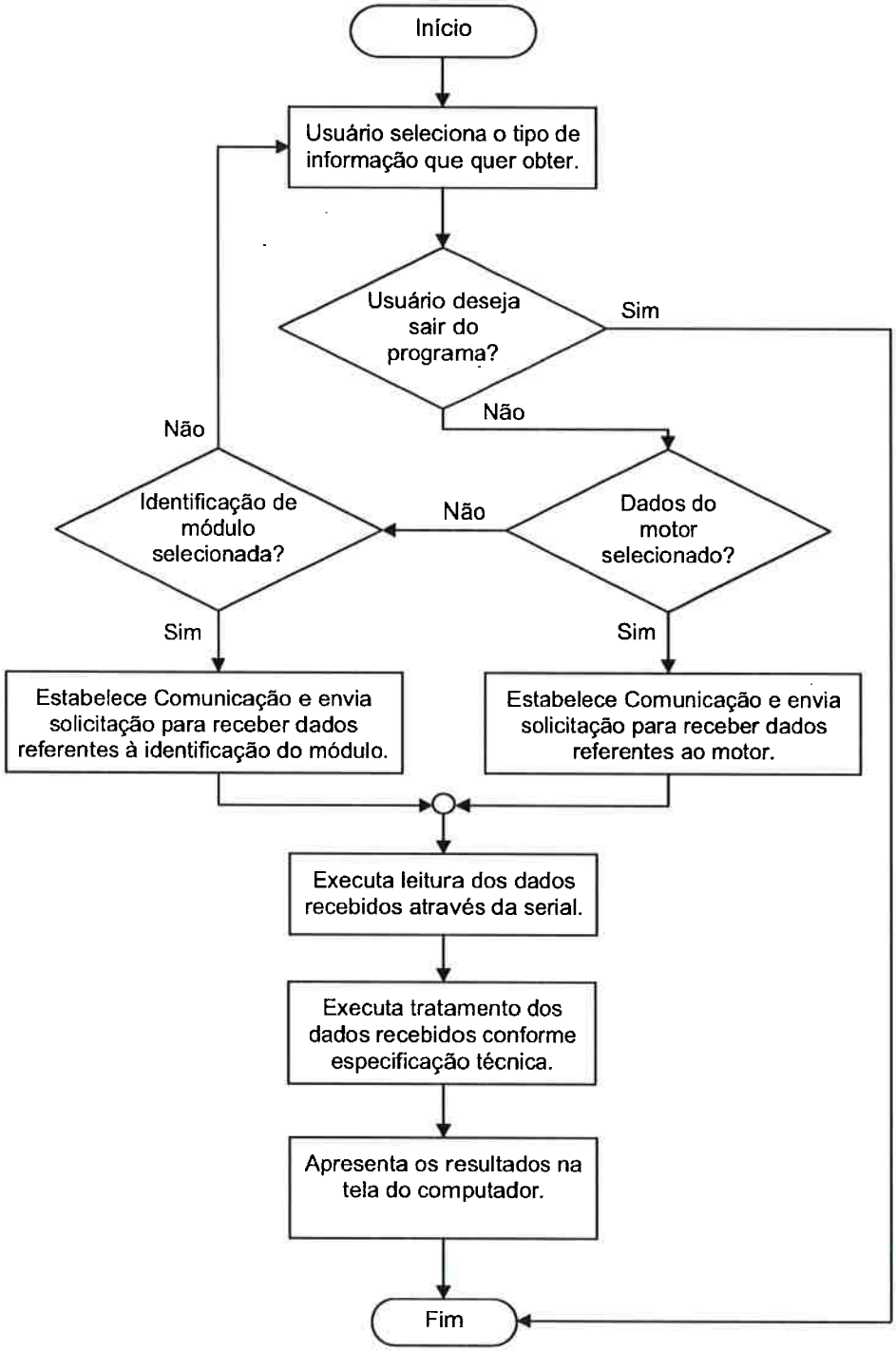


Figura 16 - Tela de leitura da identificação do controlador do motor

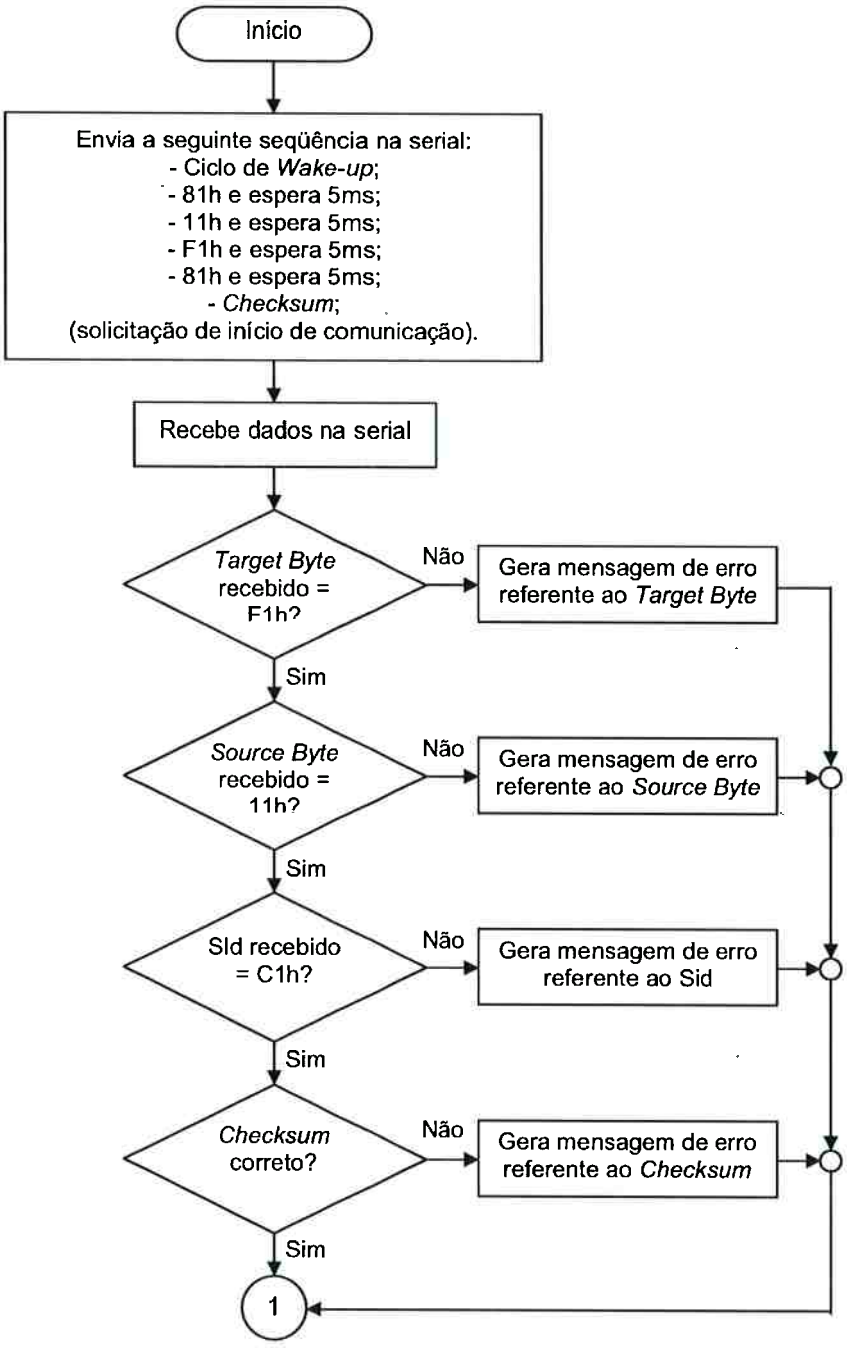
Assim, os três serviços propostos foram testados, possibilitando a leitura dos dados do motor em tempo real e a identificação do módulo controlador sempre que necessário.

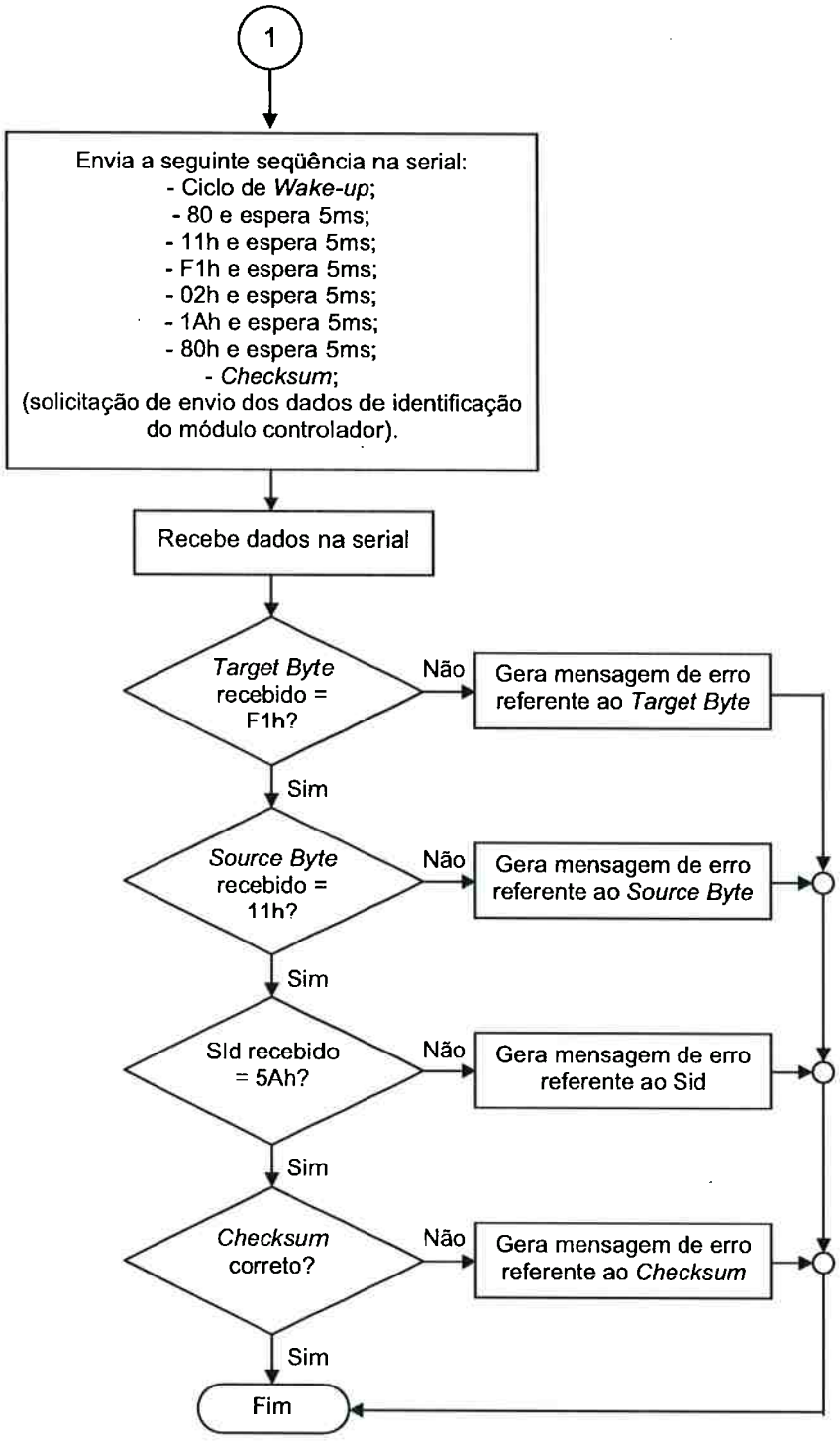
A seguir é apresentado o fluxograma do software desenvolvido:

Primeiramente é apresentada uma visão macro do funcionamento do software:

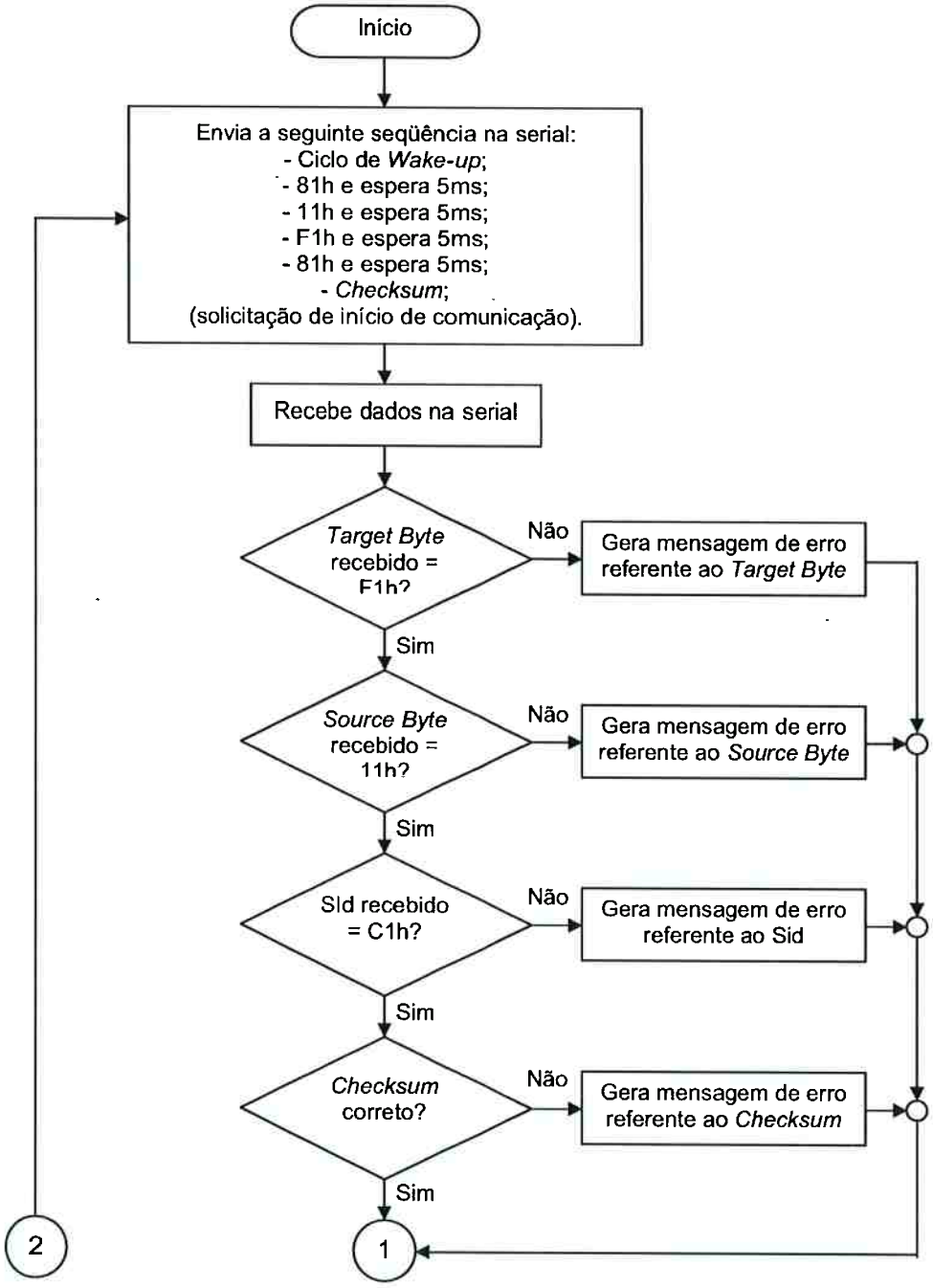


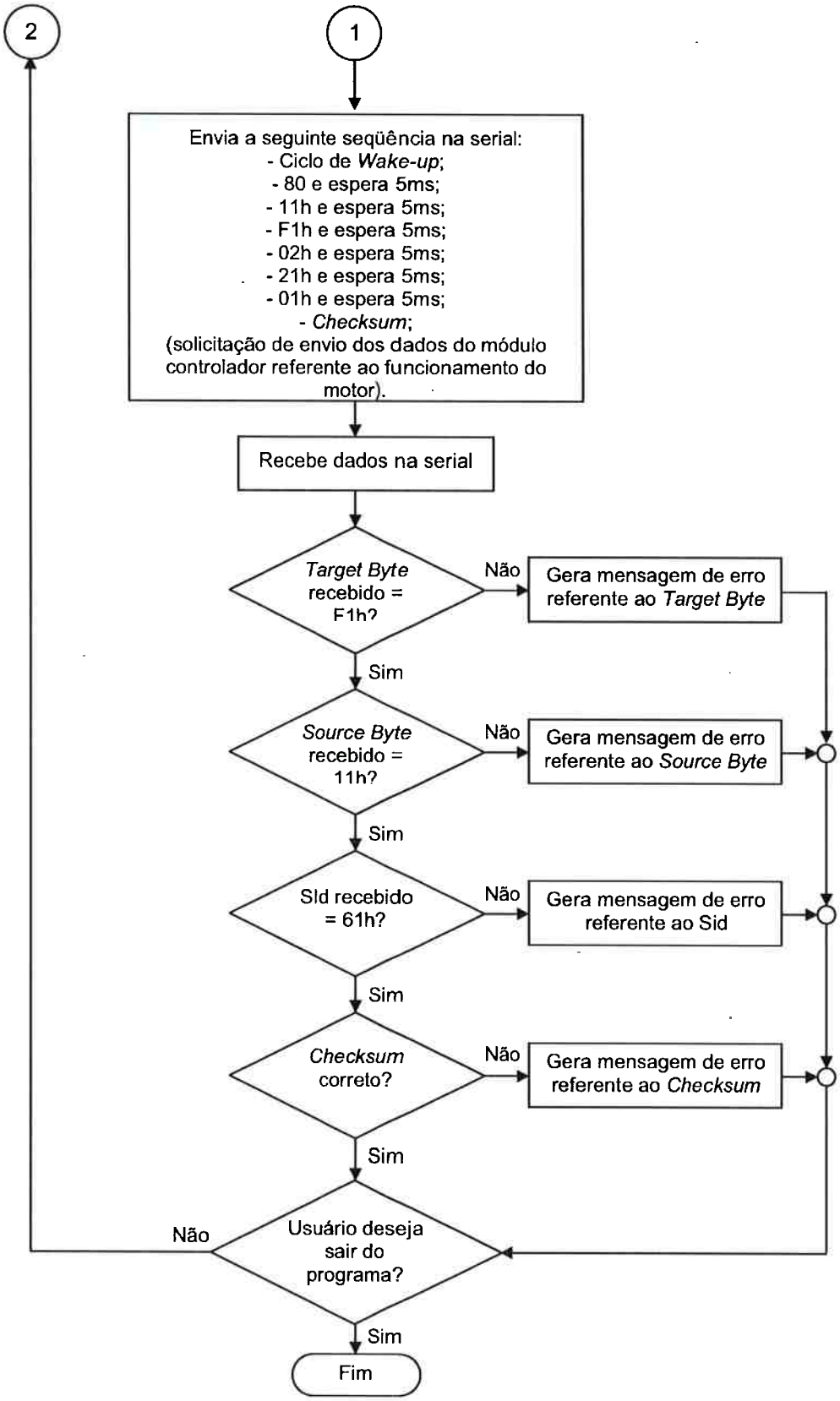
Agora é apresentado o detalhamento da função "Estabelece Comunicação e envia solicitação para receber dados referentes à identificação do módulo.":




1
Envia a seguinte seqüência na serial:
- Ciclo de *Wake-up*;
- 80 e espera 5ms;
- 11h e espera 5ms;
- F1h e espera 5ms;
- 02h e espera 5ms;
- 1Ah e espera 5ms;
- 80h e espera 5ms;
- *Checksum*;
(solicitação de envio dos dados de identificação do módulo controlador).
Recebe dados na serial
Target Byte recebido = F1h?
Não
Gera mensagem de erro referente ao *Target Byte*
Sim
Source Byte recebido = 11h?
Não
Gera mensagem de erro referente ao *Source Byte*
Sim
SId recebido = 5Ah?
Não
Gera mensagem de erro referente ao Sid
Sim
Checksum correto?
Não
Gera mensagem de erro referente ao *Checksum*
Sim
Fim

Agora é apresentado o detalhamento da função "Estabelece Comunicação e envia solicitação para receber dados referentes ao motor.":




2
1
Envia a seguinte seqüência na serial:
- Ciclo de Wake-up;
- 80 e espera 5ms;
- 11h e espera 5ms;
- F1h e espera 5ms;
- 02h e espera 5ms;
- 21h e espera 5ms;
- 01h e espera 5ms;
- Checksum;
(solicitação de envio dos dados do módulo controlador referente ao funcionamento do motor).
Recebe dados na serial
Target Byte recebido = F1h?
Não
Gera mensagem de erro referente ao Target Byte
Sim
Source Byte recebido = 11h?
Não
Gera mensagem de erro referente ao Source Byte
Sim
SId recebido = 61h?
Não
Gera mensagem de erro referente ao Sid
Sim
Checksum correto?
Não
Gera mensagem de erro referente ao Checksum
Sim
Usuário deseja sair do programa?
Não
Sim
Fim

Com o término da aplicação em "Delphi", foi possível verificar o comportamento do protocolo "KW2000" na prática. Cada comando executado, seja pelo lado da interfáce, seja pelo lado do módulo de controle do motor do veículo, respeitou integralmente as especificações detalhadas nas normas consultadas. Através de testes que simularam o descumprimento das especificações do protocolo, foi possível notar o envio de mensagens de erro por parte do módulo estudado, o que caracterizou o correto manuseio de erros, conforme descrito na parte teórica deste trabalho.

5 Conclusão

Através da execução deste trabalho foi possível notar que o protocolo "KW2000" possui características bem definidas pelas normas ISO que o padroniza.

Uma vez vencidos os desafios, alcançou-se as pré-condições necessárias para enviar e receber as informações através da aplicação (construída sobre a linguagem de programação "Delphi") e evidenciou-se o bom funcionamento do protocolo levando-se em consideração a atualização contínua dos valores apresentados na tela do *notebook*. Tais valores mostraram-se em conformidade, quando comparados com os dados lidos a partir de uma ferramenta de diagnóstico dedicada, utilizada pela rede de concessionárias de uma das maiores montadoras do país.

Com o término deste trabalho, conclui-se que é perfeitamente possível a realização da programação de ferramentas de diagnóstico em território nacional, visando reduzir drasticamente o custo relacionado a este item nas montadoras que ainda não o fazem e possibilitando o surgimento de novos postos de trabalho no Brasil. Para tanto, basta o aprendizado da linguagem de programação da ferramenta de diagnóstico e a aquisição dos softwares necessários, além, é claro, da contratação de profissionais habilitados e com perfil adequado para tal função.

Este trabalho caracteriza uma pequena porção das funções que são padronizadas pelo protocolo "KW2000". Itens como o envio de comandos para movimentar componentes (gerenciados pelo módulo de controle do motor) e a programação de variáveis (tipo de combustível, número de identificação do veículo (VIN), número da peça, etc.) são comandos que podem ser desenvolvidos a partir deste trabalho, recorrendo-se às normas que padronizam o protocolo "KW2000". Fica o convite para que futuros projetos apliquem estas funções visando incrementar a aplicação desenvolvida nesta dissertação.

6 Referências

International Organization for Standardization. ISO 14229: 1996 Road Vehicles - Diagnostic Systems Diagnostic Services Specification

_____. ISO 14230-1: 1999 Road Vehicles - Diagnostic Systems – Physical Layer

_____. ISO 14230-2: 1999 Road Vehicles - Diagnostic Systems – Data Link Layer

_____. ISO 14230-3: 1999 Road Vehicles - Diagnostic Systems – Application Layer

_____. ISO 14230-4: 2000 Road Vehicles - Diagnostic Systems – Requirements for Emission Related Systems

_____. ISO 9141-2: 1994 CARB Requirements for Interchange of Digital Information.

KEYWORD Protocol 2000 Part 2: Data Link Layer Recommended Practice

KEYWORD Protocol 2000 Part 3: Implementation Recommended Practice

Society of Automotive Engineers. SAE J1978: OBD II Scan Tool

_____. SAE J1979: 1998 Draft E/E Diagnostic Test Modes revised for ISO 14230-4