	Data	Signatures	
Author:	30/03/2020		Title: **Guia do Desenvolvedor Sistema EAR - FW**
Verif.:			
Aprov.:			

1. Escopo

O objetivo deste documento é descrever os aspectos necessários do sistema EAR para possibilitar o desenvolvimento do "firmware" da ECU EAR. É indispensável a leitura prévia da última versão do documento "EAR System Developer's HW Guide" para que o desenvolvedor de software entenda o sistema EAR e seu HW.

Estão previstas ampliações de escopo do projeto para suportar respiração assistida, Somente para os pacientes em fase de recuperação. A eletrônica do projeto EAR já contempla expansões para esta função e outras.

Este projeto pode ser alterado a qualquer momento, sem aviso prévio.

Disclaimer: não assumimos nenhuma responsabilidade sobre qualquer dispositivo construído a partir das informações deste documento e orientamos para que os dispositivos sejam projetados na estrita observância às normas locais, que sigam as recomendações de especialistas e que sejam homologados pelas agências reguladoras em todas as fases.

2. Controle do Documento

- 30/03/2020: Emissão R00 – Português BR
  - definidos requisitos gerais
  - definidos em FW01: testes de hardware e ferramentas necessárias para próximas etapas
  - definidos em FW01: testes para podermos modelar parâmetros importantes de funcionamento
  - operação com paciente a ser implementada

3. Referências

- Open Source COVID19 Medical Supplies facebook: https://www.facebook.com/groups/670932227050506/
- COVID-19 Air BRASIL - Fast production of assisted ventilation devices https://www.facebook.com/groups/235476464265909/
- RDSV200320 Emergency Automatic Respirator Specification R00 – Preliminary
- RDSV200322 EAR Hardware Proposition Draft R00 – PTBR
- **MIT E-Vent – "Key Ventilation Specifications"**
- **RDSV200329 EAR System Developer's HW Guide PTBR R00**

4. Abreviaturas utilizadas

Abreviaturas e termos utilizadas em todo o documento:
- EAR: Emergency Automotic Respirator
- FDC: Fim de Curso

- BAG: bolsa de ar, pode ser a do AMBU ou a do FIO2
- AMBU: Equipamento médico existente, utilizamos somente o BAG para comprimir ar para o paciente
- FIO2: Fraction of Inspired O2 define a proporção de O2 no ar inspirado pelo paciente
- ECU: Electronic Control Unit
- PEEP: Pressão positiva expiratória final. Para eletrônicos e SW developers, se trata de um "Offset DC" na pressão do ar do pulmão do paciente. A pressão nunca cai a menos de 5-10cm de H20.
- HEPA: tecnologia empregada em filtros de ar com alta eficiência na separação de partículas (wikipedia)
- NPN: polaridade da conexão do sensor ou atuador, comuta ou é comutado pelo GND (Low Side)
- SPI: Serial Peripheral Interface, interface serial síncrona, baseada em registradores de deslocamento
- WIP: Work in Progress, trabalho não finalizado
- GPIO: General Purpose IO, ou porta de uso geral do arduíno
- SOA: Safe operating area, ou área de operação segura do componente
- Pots: Potenciômetros
- CW: Clockwise ou sentido de compressão do AMBU (A+ B-)
- CCW: Counterclockwise ou descompressão do AMBU (A- B+)

5. Considerações Iniciais, Requisitos Básicos, Visão Global do Sistema EAR, Visão Geral do Hardware Controlado pelo Sistema EAR

Ver **RDSV200329 EAR System Developer's HW Guide PTBR R00.**

6. Requisitos do SW ECU EAR

6.1. Requisitos Gerais

6.1.1. Target HW

Qualquer firmware deve ser desenvolvido para rodar em um Arduíno Pro Mini, mediante bootloader. Embora tenha sido previsto um conector ICSP no hardware, não disponível no Arduíno Pro Mini, isto exigiria um equipamento de gravação AVR que pode não ser disponível.

- Pinout do Arduino Pro Mini:

PINO	FUNÇÃO PINO	OBSERVAÇÕES
TXO	TXO	Tx Bootloader, monitor serial ou TxD Rede
RXI	RXI	Rx Bootloader, monitor serial ou RxD Rede
RST	/RESET	Reset geral, para acionamento usar open collector NPN

GND	GND	GND
D2	GPIO	Reservado Rede_RTS
D3	GPIO	Latch Inputs expansor de IO – SH /LD 74HC165
D4	GPIO	Buzzer OUT
D5	GPIO	Motor Low Side B
D6	PWM	PWM Motor Hi Side B
D7	GPIO	Checklamp LED Vermelho
D8	GPIO	Motor Low Side A
D9	PWM	PWM Motor Hi Side A
D10	GPIO	Latch Outputs expansor de IO – RCK 74HC595 +Edge
D11	MOSI	MOSI expansor de IO
D12	MISO	MISO expansor de IO
D13	SCK	SCK expansor de IO – normalmente ligado a um LED
A0	AN0	Leitura Analógica POT1
A1	AN1	Leitura Analógica POT2
A2	AN2	Leitura Analógica POT3
A3	AN3	Leitura Analógica POT4 ou futura entrada Multiplex AN
VCC	VCC	+5V
RST	/RESET	Reset geral, para acionamento usar open collector NPN
GND	GND	GND
RAW	RAW	Não Conectado – Recomendado remover o regulador interno do Arduíno.

A7	AN7	Leitura Analógica Corrente do Motor
A6	AN6	Leitura Analógica Elemento Externo (ex: Sensor Pressão)
A5	AN5	Leitura V_Motor após PPTC
A4	AN4	Reservado

6.1.2. Dump Serial

IMPORTANTE: Todo software desenvolvido deve dumpar, via monitor serial, em Ascii e e em tempo real, os dados relevantes do sistema, tais como:

- Posições lidas dos pots
- Fase do processo respiratório
- Warnings e Erros
- Estados de máquinas de estado importantes (ME do Motor, etc)

Na inicialização devem ser reportados Tipo de FW, Versão de FW, Parms em memória não volátil, Erros, se puderem ser recuperados da RAM, ou Erros graves armazenados em NVM.

Sugerimos que esta seja a primeira coisa a ser implementada, para termos uma ferramenta de auxílio ao desenvolvimento do firmware. A taxa de transmissão deve ser a mesma utilizada para o bootloader, 57K6 bps. Esta ferramenta será doravante denominada FlexLed.

6.1.3. Tratamento de IO baseado em SPI

- o Projeto EAR tem expansão de IOs lentos, via SPI, por módulos. Cada módulo proporciona 8 entradas para sensores NPN e 8 saídas para relés com acionamento NPN. Denominamos lentos porque não serão frequentemente lidos ou atualizados, as taxas vão depender da implementação de software específica, mas penso que devem ficar no **range de 1 a 10 ms entre cada atualização. Um dos gargalos importantes é a leitura dos sensores que promovem o desligamento dos motores, quanto maior o tempo entre leituras, mais o motor irá se deslocar**. Cuidado deve ser tomado para ter bufferização de todas estas saídas em software, porque as leituras das entradas sobreescrevem, SEMPRE, as saídas. Embora as saídas necessitem de um sinal de "Latch_Outputs" para serem efetivadas, **um pequeno ruído elétrico pode promover isto.**

O tratamento de IO baseado em SPI é bastante simples, mas alguns cuidados devem ser tomados: antes de realizar as transações devemos realizar um pulso negativo em Latch_Inputs (D3) para carregar as leituras nos registradores de deslocamento do 74HC165 a serem lidos pela SPI e após as transações devemos realizar uma pulso positivo em Latch_Outputs para validar os dados transmitidos via SPI para o 74HC595.

É importante entender a CPI

- ver https://www.arduino.cc/en/reference/SPI e exemplos
- tratar a SPI em módulos de 8 bits

- lembrar que o número de entradas e saídas podem variar, sempre em módulos de 8. Assim podemos ter 8, 16, 24 ou 32 saídas e entradas. Todas as saídas tem que ser bufferizadas em software, porque as leituras das entradas sobreescrevem, SEMPRE, as saídas. Embora as saídas necessitem de um sinal de "Latch_Outputs" para serem efetivadas, **um pequeno ruído elétrico pode provocar isto.**

Exemplo, podemos ter um vetor InputsExp [4] e OutputsExp [4]. O procedimento vai ser dependente do número de placas que tivermos.

```
#include <SPI.h>
byte  InputsExp [4];
byte OutputsExp [4];
#define Latch_Inputs
#define Latch_Outputs 10     // deve ser inicializada com 0
#define Latch_Inputs 3

// antes do procedimento SPI utilizar
digitalWrite(Latch_Inputs, LOW);  // garante que houve a transferência,
realizada por nível, do HC165
// talvez precise inserir aqui um pequeno delay de alguns microssegundos
digitalWrite(Latch_Inputs, HIGH); // habilita operação HC165 como shift
register

// depois do procedimento SPI utilizar
digitalWrite(Latch_Outputs, HIGH);
```

```
// talvez precise inserir aqui um pequeno delay de alguns microssegundos
digitalWrite(Latch_Outputs, LOW);
```

procedimento para 1 placa:
SPI.beginTransaction();
InputsExp[0]=SPI.transfer(OutputsExp[0]);
SPI.endTransaction();

procedimento para 2 placas:
SPI.beginTransaction();
InputsExp[0]=SPI.transfer(OutputsExp[1]);
InputsExp[1]=SPI.transfer(OutputsExp[0]);
SPI.endTransaction();

procedimento para 3 placas:
SPI.beginTransaction();
InputsExp[0]=SPI.transfer(OutputsExp[2]);
InputsExp[1]=SPI.transfer(OutputsExp[1]);
InputsExp[2]=SPI.transfer(OutputsExp[0]);
SPI.endTransaction();

procedimento para 4 placas:
SPI.beginTransaction();
InputsExp[0]=SPI.transfer(OutputsExp[3]);
InputsExp[1]=SPI.transfer(OutputsExp[2]);
InputsExp[2]=SPI.transfer(OutputsExp[1]);
InputsExp[3]=SPI.transfer(OutputsExp[0]);
SPI.endTransaction();

6.2. Requisitos para o FW01 - Teste de Hardware

O objetivo do FW01 é possibilitar um teste simples para verificar se o hardware da EAR ECU é funcional.
Devem ser verificados:
- funcionamento e excursão dos pots (verificação via FlexLed está OK)
- funcionamento dos módulos de expansão de IO, mediante testes com loopback** de entradas e saídas. **loopback significa: IN1A ligado a OUT1-1, IN1B ligado a OUT11-2 e assim por diante. Lembrar que se foi montado um ULN2003 não haverá um pino de saída em OUT1-1 e o software deve reportar a falha.
- funcionamento do driver de motor, de acordo com o tipo de driver montado: relé ou mosfet.

6.2.1. Testes dos Pots

- basta realizar a leitura dos pots e reportar no FlexLed algo do tipo P1xx P2yy P3zz e P4ww onde xx, yy, zz e ww podem ser as leituras analógicas dos pots dos 8 bits mais significativos do ADC, representadas em hexadecimal.

6.2.2. Funcionamento dos módulos de expansão de IO

- basta realizar escritas e leituras dos IOs baseados em SPI e reportar no FlexLed o resultado.
Uma sequência de testes exemplo nesta primeira fase poderia ser algo do tipo:

Tx00000001 Rx00000001
Tx00000010 Rx00000010
...
Tx10000000 Rx10000000

Os resultados podem ser apresentados de forma concomitante com resultados de leituras dos pots anteriores, algo tipo:
P1xx P2yy P3zz P4ww Tx10000000 Rx10000000

Pode parecer algo inútil, mas será útil para desenvolvimento, testes de produção e campo.

6.2.3. Funcionamento do Driver de Motor

Este teste consiste em realizar sequências de testes do motor que permitam avaliar o driver e o motor, um de cada vez.

O teste do driver é realizado com um motor OK e o teste do motor é realizado com um driver já validado.

Os pots tem os seguinte significado:
P1 define disparos por minuto, na faixa de 8 a 30
P2 define o PWM aplicado na faixa de 30 a 100%, começar com mínimo
P3 define tempo de acionamento CW e CCW

Obs: não há fim de curso por software, mas pode ser implementado por hardware se estiver ligado na alimentação do motor. Uma sugestão é hackear o amplificador de corrente, provocando parada por percepção de curto.

No caso de ser um driver de relés P2 não vai influenciar nada, dev ser realizada alteração de corrente mediante regulagem de tensão da fonte específica do motor, mandatória no caso de relés.

Estes testes podem ser utilizados como testes “Open Loop” para ajudar o mapeamento de pressões de funcionamento e outros necessários para a implementação do sistema.

O FlexLed é da etapa anterior acrescido da etapa do motor e da corrente envolvida, reportado como Mxx Iyy, ficando assim:
Mxx P1xx P2yy P3zz P4ww Tx10000000 Rx10000000 Iyy

A sequência do Motor xx reportada em Mxx é a seguinte:
00: motor em repouso, aguardando start (=motor stop CCW)
01: motor start, movimento CW - Clockwise ou sentido de compressão do AMBU (A+ B-)
02: motor regime CW

03: motor final CW
04: motor stop CW
05: motor start, movimento CCW - Counterclockwise ou descompressão do AMBU (A- B+)
06: motor regime CCW
07: motor final CCW

O valor de PWM em cada etapa Mxx deve ser parametrizável. Isto será necessário para o mapeamento.

A corrente do motor Iyy, yy é obtido da mesma forma que a leitura dos pots, se trata dos 8 bits mais significativos da leitura do amplificador de corrente do motor. Correntes acima de um threshold programável (ex: 3 amperes terá Vadc correspondente a aproximadamente 1,29V) devem ser reportados por uma piscada de 200ms no led vermelho em D7.