CONTROLADOR DE CÓDIGO COMUM UTILIZADO EM CONFIGURAÇÕES E PROGRAMAÇÃO DE DISPOSITIVOS INDUSTRIAIS

Gilson Fonseca Peres Filho, gilsonfonseca@yahoo.com.br[1]
Marcio José da Cunha, cunhamjg@gmail.com[1]
Carlos Augusto Bissochi Junior, bissochi.jr@gmail.com[1]
Josué Silva de Morais, josuemorais@yahoo.com.br[1]
Fábio Vincenzi Romualdo da Silva, fabiovince@gmail.com[1]

[1] Universidade Federal de Uberlândia – UFU, Faculdade de Engenharia Elétrica- FEELT, Núcleo de Pesquisa em Controle e Automação – NPCA, Av. João Naves de Ávila, 2121 Bloco 3N,Uberlândia, MG, Brasil

Resumo: *Atualmente, existem diversos protocolos de comunicação utilizados em redes industriais. Esta diversidade dificulta a interconexão entre redes e a configuração de sistemas supervisórios, CLPs, transmissores, IHMs, dentre outros dispositivos de acionamento e controle industriais. Deste modo, cada fabricante desenvolve e disponibiliza seu próprio configurador/interface de programação de CLPs, transmissores e IHMs. Esta situação dificulta a formação técnica especializada no sentido de realizar manutenção e expansão de processos industriais. Assim, a criação de recursos computacionais de código livre que facilitem a fabricantes e pesquisadores a construção de configuradores padronizados, propiciaria redução de investimento por parte do fabricante e melhores condições de formação técnica especializada. Além disso, as especificações de alguns padrões abertos são muito caras, extensas, e difíceis de serem implementadas. Em relação à produtividade há algo errado, pois um time de desenvolvimento em cada companhia lê, interpreta, testa os equipamentos e, posteriormente, as fundações testam e validam aqueles equipamentos para homologação. Os configuradores e equipamentos atuais disponibilizam somente recursos para a configuração de controles tradicionais do tipo PID. Este é mais um fato que motiva a criação de configuradores e equipamentos que permitam a implementação fácil de controles mais eficientes. Pois, melhorias no controle podem representar lucros expressivos no processo de fabricação e, dependendo do processo, grande redução de impacto ambiental. Em consideração as vantagens mencionadas anteriormente, este trabalho propõe a criação de funções com o objetivo de padronizar o processo de configuração e programação de dispositivos industriais. As funções são vinculadas a uma máquina virtual projetada para sistemas embarcados com baixo custo computacional. Este conjunto de funções, cujo código será aberto ao final do trabalho, permitirá a fabricantes e pesquisadores criarem interfaces de configuração e programação padronizadas, com a vantagem adicional da implementação de algoritmos de controle personalizados.*

Palavras-chave: *padronização, código aberto, algoritmos personalizados, sistemas de controle, redução de custos, formação técnica, pesquisa, desenvolvimento*

1. INTRODUÇÃO

Atualmente, há um grande número de padrões abertos de tecnologias para automação como Profibus, Fieldbus Foundation (FF), DeviceNet, EtherNet/IP, dentre outros (Gaj *et al.*, 2013; Sauter, 2010; Shahraeini et al., 2011). Vale salientar que, no ramo de automação industrial, entende-se por "padrões abertos", aqueles que podem ser comprados e implementados por qualquer fabricante de dispositivos industriais.

A aceitação de padrões abertos fizeram com que palavras como interoperabilidade, intercambialidade e unificado se tornassem comuns entre engenheiros e técnicos de instrumentação. A interoperabilidade entre equipamentos consiste na comunicação entre equipamentos de fabricantes diferentes, desde que utilizem o mesmo padrão. Os dispositivos tornaram-se intercambiáveis porque um dispositivo de um determinado fabricante pode ser substituído por outro de fabricante diferente. E o termo unificado refere-se a possibilidade de se realizar comunicação entre redes com protocolos diferentes por meio de *gateways* (Panton, *et al.*, 2007).

A maioria dos fabricantes, aproveitaram as benesses deste novo mundo e começaram a desenvolver equipamentos de acordo com estes padrões.

No entanto, para o fabricante, existem alguns custos inerentes aos padrões abertos que são desconhecidos aos usuários. A forma como esses padrões foram projetados é uma fraqueza e um desafio ao mesmo tempo. Isto acontece porque as especificações de alguns padrões são muito caras, extensas, difíceis de serem implementadas e propensas a ambiguidade. Desse modo, existe um grupo de pessoas em cada fundação que são responsáveis por desenvolver e dar

manutenção em seus respectivos padrões. Estes grupos gastam muito tempo escrevendo, revisando, melhorando e frequentemente arbitrando a interpretação do próprio texto para resolverem contestações e problemas. Por outro lado, os grupos de desenvolvimento em cada fábrica, interpretam e testam inúmeras vezes os equipamentos desenvolvidos. Mas antes que um equipamento seja produzido e comercializado, é necessário, para a maioria dos padrões, que seja testado e validado pela respectiva fundação ou organização.

Apesar da "padronização" de tecnologias, existem particularidades referentes a configuração e programação dos dispositivos industriais. Não existe, portanto, portabilidade de programação e configuração entre os diversos dispositivos industriais construídos por fabricantes diferentes. Esta situação dificulta a formação técnica especializada no sentido de realizar manutenção e expansão de processos industriais e contribui para que o usuário torne-se refém da tecnologia adquirida.

Há um segundo tópico a ser analisado que é a complexidade dos sistemas atuais. Talvez a arquitetura de controle de processos distribuído mais avançada seja a da Fieldbus FOUNDATION ™ (FF). Ela promete trazer controle ao campo, prover informações de qualidade dos sinais e um sistema poderoso de alarme e gerenciamento, modos de operação e determinismo de rede. Um sonho que se tornou realidade!

A FF criou o modelo de bloco funcional para os processos contínuos e padronizou a camada de aplicação. Para tornar a vida dos usuários mais fácil, muitos blocos foram padronizados. Como AI, AO, DI, DO, MDI, MDO, etc (Foundation, 2008).

Ultimamente, até os transdutores, como os posicionadores de válvula, por exemplo, entraram no processo de padronização. Isto foi muito bom do ponto de vista do usuário, mas colocou mais pressão nas ferramentas de teste como ITS (*Interoperability Test Systems*), que são usadas no ITK (*Interoperability Test Kit*) (Mitschke *et al.* 1999). Esta ferramenta realiza vários testes, porém, não é capaz de testar tudo, devido ao grande número de combinações. Em cada nova versão do ITK, os fabricantes são surpreendidos por um novo teste que não era feito anteriormente ou que foi demandado devido a novas características na especificação. Geralmente, um ou dois meses são necessários para resolver todos os problemas de *bugs* no *firmware* do equipamento ou no software do ITS.

Ademais, embora os blocos funcionais flexíveis no padrão FF existam para integrar a lógica *ladder* ou qualquer outra linguagem proprietária do fabricante, permitindo alguma customização de algoritmos, eles são limitados em capacidade e velocidade para transmissores mais simples. Sendo mais capazes em controladores mais poderosos como *linking devices* e PLCs.

A partir do exposto, verifica-se o grande esforço realizado por parte das organizações e fabricantes no sentido de aperfeiçoarem os equipamentos industriais. Neste sentido, este trabalho propõe uma nova abordagem de hardware e software com o propósito de se obter os seguintes benefícios: 1) tornar a programação e configuração de dispositivos industriais portável; 2) reduzir o esforço de interpretação de padrões e de testes por parte dos fabricantes; 3) diminuir o custo do hardware do dispositivo; 4) possibilitar a implementação de controles diversos.

Para atingir estas metas, o hardware proposto pode ser definido como controladores de processos com seus próprios sensores, atuadores e qualquer outra interface física necessária. Por controladores de processo entende-se uma entidade de execução que permitiria rodar códigos arbitrários.

Considerando-se que a linguagem de desenvolvimento padrão para sistemas embarcados ainda é C. Qualquer arquitetura de hardware, provavelmente terá um compilador ANSI C compatível. Há inúmeros softwares de sucesso de código fonte aberto e desenvolvimento compartilhado rodando atualmente, com crescente contribuição de vários programadores em todo mundo. Os softwares abertos mais conhecidos atualmente são: Linux (kernel e coletânea, isto é, coletânea de outros softwares que fazem o sistema), Apache e diversos softwares GNU.

Assim, uma solução de código aberto reduziria muito os problemas de interoperabilidade, interpretação equivocada da norma e o desenvolvimento poderia concentrar na parte específica do equipamento ao invés do padrão.

Outro apelo ao software aberto seria a contribuição de entidades educacionais. Para a maioria delas é praticamente impossível contribuir com os padrões que são fechados e arcar com os custos de anuidade requeridas dos membros.

Todos os recursos requeridos por uma dada aplicação devem ser carregados dinamicamente no equipamento. Um conjunto padrão de recursos, além do ambiente de execução deve ser padronizado, como funções matemáticas, acesso de I/O, temporizadores e interfaces de protocolo.

Quaisquer recursos, como alarmes, qualidade de sinal, registro de histórico, conjunto de variáveis (*views*), ligações lógicas entre equipamentos (*links*), etc, fica a cargo da ferramenta de configuração. Ela é responsável por gerar um código comum a partir de qualquer linguagem de programação como *ladder*, diagrama de blocos, texto estruturado, dentre outras linguagens. Tal ferramenta permite ainda que o usuário edite o código gerado antes de aplicá-lo. Deste modo, um novo mercado para ferramentas de configuração intercambiáveis e capacidades avançadas também poderá ser criado. Vale salientar, no entanto, que esta proposta não tem a pretensão de "forçar" a substituição da tecnologia já implementada nos dispositivos industriais. Ela visa contribuir para o desenvolvimento da área em questão. Por fim, pretende-se, ao final do projeto, disponibilizar um código aberto. Permitindo, assim, a pesquisadores e empresários realizarem pesquisa e desenvolvimento de dispositivos industriais.

2. NÚCLEO DO DISPOSITIVO PROPOSTO E EXEMPLO DE APLICAÇÃO

A Figura 1 apresenta o Núcleo do Dispositivo Proposto (NDP) e um exemplo de aplicação no meio industrial. Dentro do núcleo, encontra-se o Diagrama de Blocos do Software Proposto (DBSP), que é composto pelos blocos:

LuaVM, Lua Manager, HAL *(Hardware Abstraction Layer)*, Net Services, Net port e OS port. Estes blocos fazem parte do código comum que deve ser implementado para se obter os benefícios propostos.

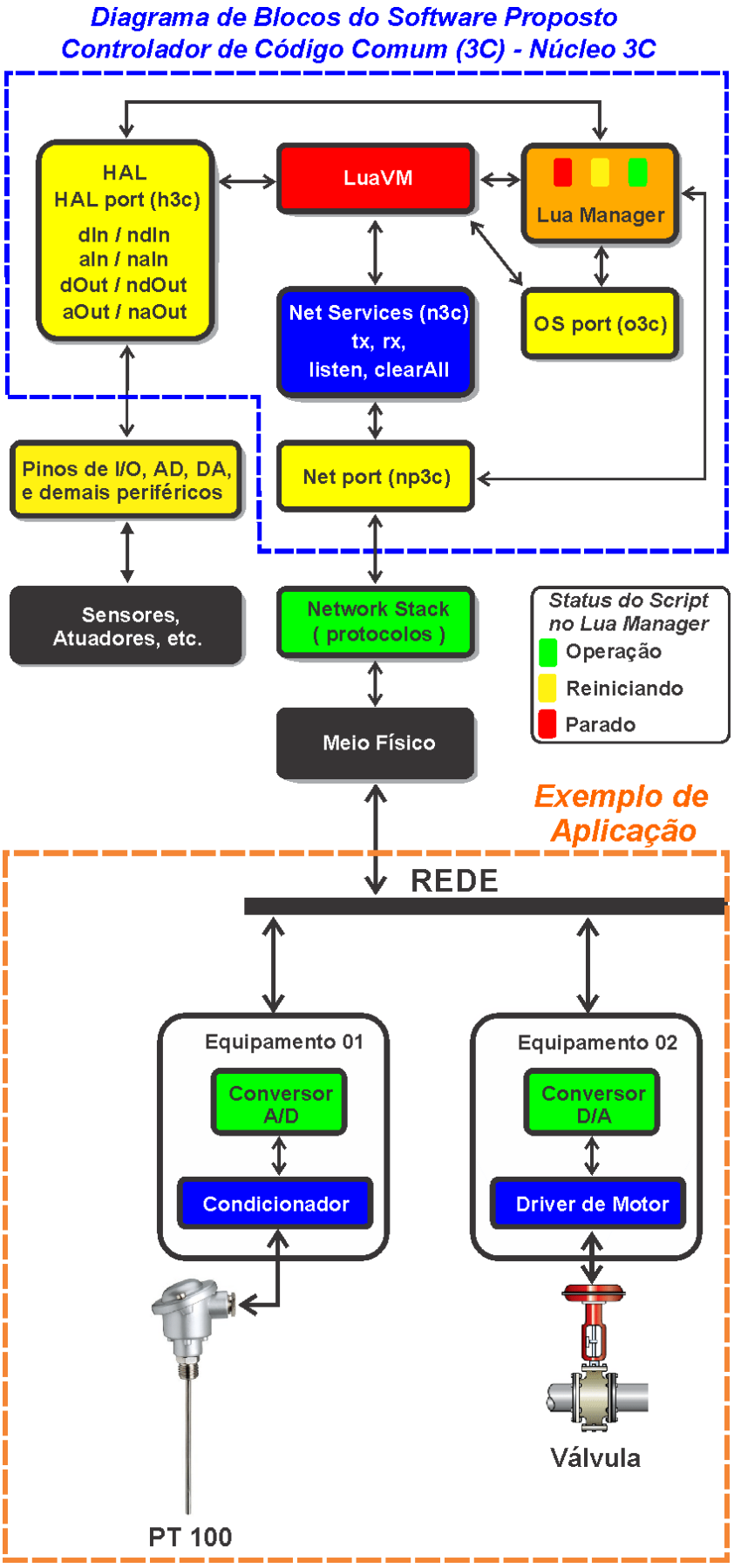

Figura 1. Núcleo do Dispositivo Proposto e Exemplo de Aplicação.

LuaVM – Este bloco representa a máquina virtual Lua, que permite a execução de scripts Lua. Vale ressaltar que, Lua é uma linguagem de programação simples, eficiente, portável e leve, como apresentado em Ierusalimschy *et al*. (2001). Diversas características da linguagem Lua foram motivadas pela indústria e dicas de usuários. Por isso, atualmente, ela é largamente utilizada na indústria de jogos no mundo todo. As principais características da máquina Lua que motivaram a sua utilização nesta proposta foram:

- **Simplicidade:** Desde o início, esta linguagem foi projetada para ser simples. Isto implica uma sintaxe com um número reduzido de construtores e de modo que pudesse ser codificada em C;
- **Eficiência:** É uma linguagem que possui um compilador eficiente, rápido, de apenas uma passagem e que possui uma máquina virtual veloz.
- **Portabilidade:** Lua foi projetada para ser executada no maior número de plataformas possível. A intenção de seus desenvolvedores foi a de desenvolver o núcleo Lua para rodar em qualquer plataforma, sem a necessidade de sofrer modificações
- **Leveza:** Até o momento, para o Linux, a versão completa do interpretador Lua, com todas as bibliotecas padrão, ocupam menos de 150 Kbytes; sendo que o núcleo ocupa menos de 100 Kbytes. Isto, contribui para que Lua seja uma das linguagens mais rápidas no campo das linguagens de script, isto é, no ramo das linguagens interpretadas e dinamicamente tipadas.

Devido as características apresentadas, todo equipamento que for implementado com uma máquina virtual Lua, poderá executar um script Lua. Desse modo, se os equipamentos industriais possuíssem uma LuaVM, poderiam executar o script Lua de outro dispositivo que fosse fabricado para executar funções semelhantes. Por exemplo: Um script Lua, elaborado para um CLP de um determinado fabricante, poderia ser executado por um CLP de outro fabricante que possuísse os mesmos recursos de hardware.

Portanto, nesta proposta de trabalho, a LuaVM proporciona portabilidade na programação e configuração dos dispositivos que serão implementados com o Controlador de Código Comum (3C) proposto.

Existem alguns riscos inerentes ao uso de linguagens que usam **gc** (*garbage collector)* em sistemas embarcados. É muito difícil, se não impossível, garantir que o **gc** não entre em momentos críticos ou que não seja necessário frequentemente para liberação de recursos. Em (Klotzbuecher, *et al.*, 2011) são propostas algumas técnicas para reduzir o tempo de atuação do **gc** e tentar tornar sua execução um pouco mais determinada, atribuem ainda com grande importância o uso de dados experimentais com coleta de tempo de execução em vários cenários de uso. "Charilaos Ademais, (Kalogirou, *et al.*, 2014) propõe uma alteração no **gc** do Lua para ser executado de forma quase previsível. Do ponto de vista do script Lua o uso de variáveis locais em um loop principal com poucas chamadas de funções que tenham variáveis locais também contribuiu para a redução da necessidade do **gc** como informado em (Wilson, *et al.*, 1993).

Lua Manager - gerencia a LuaVM permitindo diversos serviços através de um protocolo, como apresentado a seguir:

- Descarregar um script na LuaVM;
- Iniciar a LuaVM;
- Parar a máquina LuaVM;
- Monitorar o I/O;
- Colocar o I/O em modo automático (controlado pelo script Lua) ou em modo manual para forçar condições de segurança.

HAL - controle de acesso ao hardware. Provê recursos para que o Lua Manager monitore o I/O, assuma o controle das saídas ou permita o acesso pela LuaVM. A princípio, por simplicidade, quatro variáveis e quatro funções serão implementadas (visíveis ao script):

- **h3c.naIn** – variável que contém o número de entradas analógicas disponíveis no hardware;
- **h3c.naOut** – variável que contém o número de saídas analógicas disponíveis no hardware;
- **h3c.ndIn** – variável que contém o número de entradas digitais disponíveis no hardware;
- **h3c.ndOut** – variável que contém o número de saídas digitais disponíveis no hardware;
- **h3c.aIn(i)** – lê uma entrada analógica. Sendo **i** [0...**h3c.naIn**] a entrada;
- **h3c.aOut(i, v)** – escreve em uma saída analógica. Sendo **i** [0...**h3c.naOut**] a saída e **v** o valor;
- **h3c.dIn(i)** – lê uma entrada digital. Sendo **i** [0...**h3c.ndIn**] a entrada;
- **h3c.dOut(i, v)** – escreve em uma saída digital. Sendo **i** [0...**h3c.ndOut**] a saída e **v** o valor;

Net services (n3c) – provê a LuaVM serviços de acesso a rede. A princípio, por simplicidade, quatro funções serão implementadas:

- **n3c.tx(id, data)** – enviar dados para a rede (id=identificador, data=dados a serem enviados). Os dados enviados são "bufferizados" e transmitidos quando possível, segundo o protocolo sobre o qual foi implementado;

- **n3c.listen(id)** – registra um id para ser recebido pelo Net Services. Uma vez registrado um id o sistema fará *buffering* de dados recebidos com aquele id;
- **n3c.clearAll()** – limpa todos os id's registrados;
- **r,id,data=n3c.rx()** – recebe qualquer dado com um dos id's registrados. Esta função retorna três parâmetros, sendo **r** true se um novo dado foi recebido, **id** o id da informação recebida, **data** os dados recebidos.

OS port (o3c) – provê recursos do sistema operacional como temporização, por exemplo. Alguns recursos fundamentais de temporização foram implementados:

- **o3c.dms(i)** – "pausa" a execução por **i** milissegundos (equivalentes a funções típicas como delay e sleep);
- **o3c.dus(i)** – "pausa" a execução por **i** microssegundos (equivalentes a funções típicas como delay e sleep);
- **o3c.ems()** – obtém o número de milissegundos do freeruning de hardware ou sistema operacional (equivalente a funções típicas como GetTickCount, getTicks, etc).

Net port (np3c) – camada que permite ao Net services e ao LuaManager acesso à rede. É por meio desta camada que é possível controlar remotamente o LuaManager para realizar a configuração e atualização do script que a LuaVM irá executar.

Esta abordagem permite que uma rede com distribuição de mensagens identificadas seja implementada sobre diversos protocolos e ao mesmo tempo propicia facilidade de uso por parte do script.

Para validação utilizou-se uma rede CAN. Nela foi atribuído o id ao id da mensagem CAN e o dado ao campo de dados da mensagem CAN que pode ter até 8 bytes.

Com o objetivo de elucidar o trabalho proposto, a seguir são apresentados alguns exemplos experimentais de aplicação do 3C.

3. RESULTADOS EXPERIMENTAIS

Cada resultado experimental apresentado a seguir é composto pelo diagrama de blocos do hardware implementado, do script utilizado em cada dispositivo e das respectivas formas de ondas obtidas por meio de um osciloscópio Agilent DSO-X 2002A. Os experimentos foram realizados com dois kits Luminary Micro - EKI-LM3S8962, que foram equipados com microcontrolador Stellaris ARM® Cortex™-M3 (50MHz de clock, 64KB de SRAM e 256KB de memória flash) e interface de rede CAN.

O consumo de memória da máquina virtual Lua + Sistema Implementando, na arquitetura ARM Cortex-M3, utilizando-se o compilador GCC 4.8.1 sem otimização é de 131kB de FLASH e 20kB de SRAM. Ademais, o consumo de memória do sistema total, incluindo RTOS, libC, etc, é de 201kB de FLASH e 35kB de SRAM.

3.1 – Exemplo de leitura, escrita e transmissão digital entre dois dispositivos

A Figura 2 mostra o diagrama de blocos do Experimento 01. Neste diagrama, o Dispositivo 01 está executando o Script A, conforme apresentado na Tabela 1 e o Dispositivo 02 está executando o Script B, mostrado na Tabela 2.

O Script A gera uma forma de onda de 500Hz no pino 0, que foi configurado pelo desenvolvedor para ser o pino PC7 do microcontrolador LM3S8962. O Script B lê o sinal transmitido na entrada 1, pino PC5 do microcontrolador, e reproduz a leitura na saída 0, que corresponde ao pino PC7 do microcontrolador. Os resultados experimentais deste experimento são exibidos na Figura 3.

Vale ressaltar que, para cada equipamento, o fabricante deverá mapear as funções do HAL aos periféricos disponibilizados no seu equipamento e apresentar uma tabela no manual de forma que o usuário saiba qual a entrada ou saída lógica está associada a uma dada entrada ou saída física.

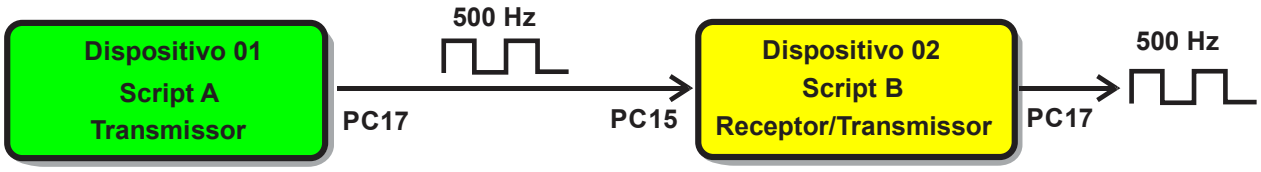

Figura 2. Diagrama de Blocos do Experimento 01.

Tabela 1. Script A - Gera onda quadrada com frequência de 500Hz na primeira saída digital.

```
while true do
      h3c.dOut(17, 1) -- coloca a saída em nível alto
      o3c.dms(1)      -- aguarda 1ms
      h3c.dOut(17, 0) -- coloca a saída em nível baixo
      o3c.dms(1)      -- aguarda 1 ms
end
```

Tabela 2. Script B - Reproduz o sinal de uma entrada em uma saída.

```
while true do
      h3c.dOut(17, h3c.dIn(15)) -- lê o valor da entrada 1 e aplica na saída 0
end
```

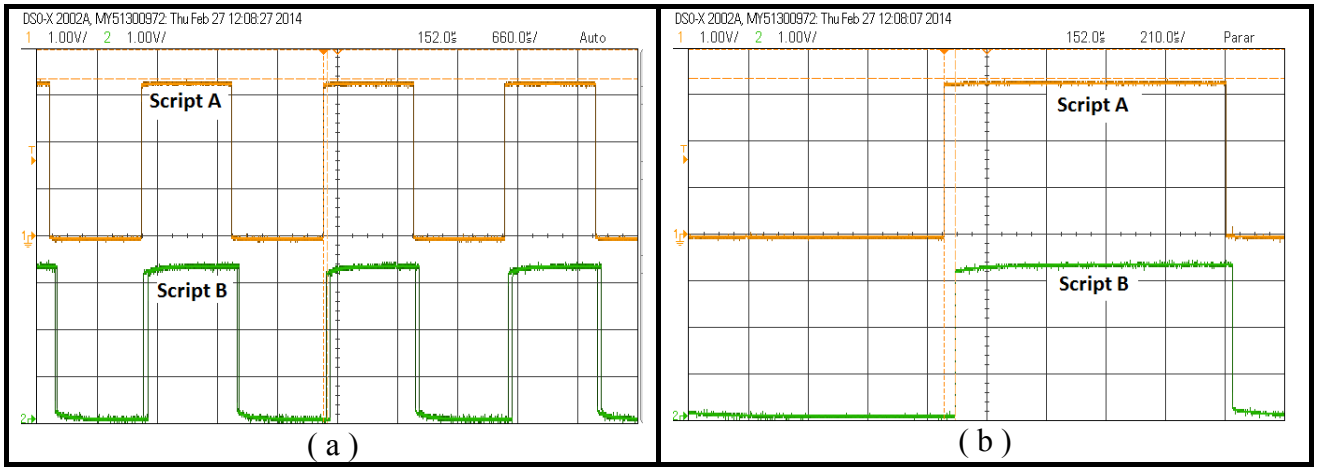

Figura 3. (a) Forma de onda de saída do Dispositivo 01 -Script A - 1V/div - 660µS/div e forma de onda de saída do Dispositivo 02 –Script B – 1V/div - 210µS/div (b) atraso de propagação entre as formas de onda de 40 µS.

3.2 – Exemplo de leitura digital, transmissão pela rede CAN e transmissão digital no outro dispositivo

A Figura 4 exibe o diagrama de blocos do Experimento 02. Neste diagrama, o Dispositivo 01 está executando o Script C, conforme mostrado na Tabela 3 e o Dispositivo 02 está executando o Script D, apresentado na Tabela 4.

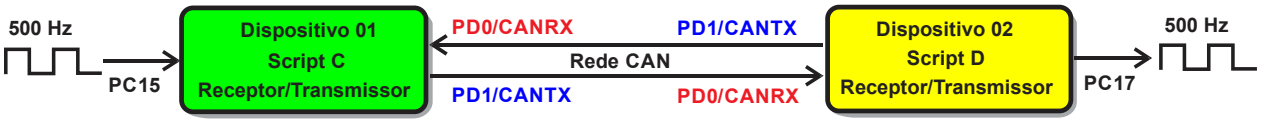

Figura 4. Diagrama de Blocos do Experimento 02.

Tabela 3. Script C – Lê sinal digital na entrada digital 15 e transmite o sinal pela rede CAN.

```
id = 10
while true do
    n3c.tx(id, h3c.dIn(15)) -- lê entrada 1 e publica na rede com id=10
end
```

Tabela 4. Script D – Recebe sinal pela rede CAN e reproduz o sinal na saída digital 17.

```
idl = 10
n3c.listen(idl) -- registra o id=10 para ser lido pelo stack de rede
while true do
     r,id,data = n3c.rx() -- lê o dado que estiver disponível (r informa se há algum)
     if r==true then
          h3c.dOut(17, data) --coloca na saída 0 o valor recebido
     end
end
```

O Script C lê o valor imposto por uma forma de onda quadrada de 500Hz no pino 15 do Dispositivo 01, que foi configurado pelo desenvolvedor para ser o pino PC7 do microcontrolador LM3S8962. Em seguida, o valor lido é publicado na rede CAN sob o id=10. Na sequência, o dado é recebido pela interface CAN do Dispositivo 02 e aplicado a saída PC17 do mesmo.

No Experimento 01, o atraso de propagação foi de 40µS. Por outro lado, no Experimento 02, quando o dado foi transmitido de um dispositivo para outro, por meio da rede CAN, o atraso de propagação sofreu um acréscimo de 780µS. Este atraso foi imposto pela rede CAN, pelas interfaces de software e pela velocidade de execução do script. Devido a rede CAN temos que uma mensagem de 4 bytes de dados como esta pode ter, no pior dos casos 158 bits, devido ao bit stuffing. Isto equivale a aproximadamente 191us para uma rede CAN com taxa de 500kbps sem levar em conta possíveis perdas de arbitragem.

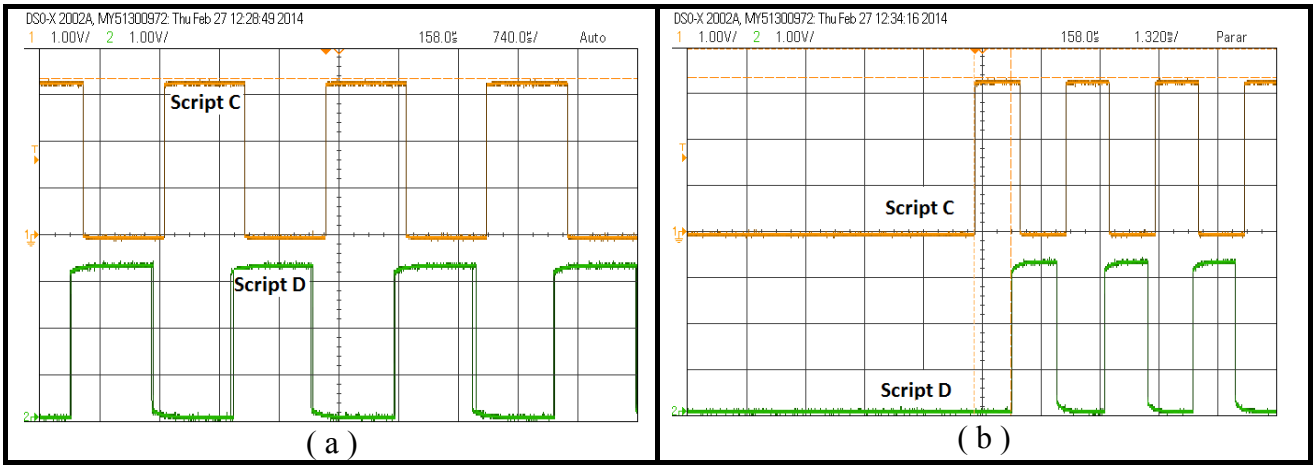

Figura 5. (a) Forma de onda de saída do Dispositivo 01-Script C-1V/div - 740µS/div e forma de onda de saída do Dispositivo 02–Script D–1V/div – 1320mS/div (b) atraso de propagação entre as formas de onda de 820 µS.

4. CONCLUSÃO

Os experimentos realizados com o Núcleo 3C proposto comprovaram que ele opera de acordo com o esperado. Isto é, a LuaVM está executando os scripts corretamente e acessando as funções do HAL, Net Services, OS port e LuaManager conforme previsto.

Embora os experimentos aqui apresentados tenham focado em scripts mais simples, com o propósito de validar a ideia, funções matemáticas mais avançadas também estão disponíveis, permitindo a implementação de algoritmos usuais como PID, Timers, Contadores, etc.

Neste sentido, o núcleo 3C proposto irá reduzir os problemas de interoperabilidade, interpretação equivocada de normas e permitir que o desenvolvedor concentre-se na parte específica do equipamento ao invés do padrão.

Ademais, este núcleo permitirá a implementação de vários dispositivos industriais virtuais, por meio do correto modelamento dos mesmos, fomentando ainda a contribuição de empresas e entidades educacionais no desenvolvimento de novas estratégias de controle, por meio de algoritmos dedicados de fácil implementação.

Finalmente, esta solução embora interessante não é capaz de resolver todos os problemas. Processos rápidos de batelada podem não suportar variações de atraso devido a um **gc** (*garbage collector)*. O foco de aplicação deste trabalho são controles contínuos, notadamente mais lentos (da ordem de 50 a 500ms) e com menor impacto devido a variações de tempo de execução. Como toda solução de engenharia há um ganho e uma perda. Neste caso, o ganho é a flexibilidade e portabilidade da solução, a perda é a provável variação do tempo de execução.

5. AGRADECIMENTOS

Os autores gostariam de agradecer ao CNPq, à CAPES e a FAPEMIG pelo apoio e suporte financeiro.

6. REFERÊNCIAS

Gaj, P., Jasperneite, J., and Felser, M., 2013, “Computer Communication Within Industrial Distributed Environment”, IEEE Transactions on Industrial Informatics, Vol. 9, No. 1, pp. 182-189.

Foundation, F., 2008, “FOUNDATION™ Specification Function Block Application Process”, Fieldbus Foundation - Part 4, Document: FF-893, Revision: FS 1.2.

Ierusalimschy, R., de Figueiredo, L. H., and Celes, W., 2001, “The evolution of an extension language: A history of Lua”, In Proceedings of V Brazilian Symposium on Programming Languages, pp. B 14-28.

Kalogirou, C., “Predictable garbage collection with Lua”, Disponível em: http://www.altdevblogaday.com/2011/07/23/predictable-garbage-collection-with-lua/ Data de acesso: 23/04/2014.

Klotzbuecher, M. and Bruyninckx, H., 2011, "Hard real-time Control and Coordination of Robot Tasks using Lua", 13th Real-Time Linux Workshop from October 20 to 22 at the Faculty of Electrical Engineering", Czech Technical University in Prague, pp. 1-7.

Mitschke, S. and Vreeland, S., 1999, "Practical Importance of the FOUNDATIONTM Fieldbus Interoperability Test System", Instrument Society of America, pp. 1-9.

Panton, R. P., Torrisi, N. and Brandão, D., 2007, "An open and non-proprietary device description for fieldbus devices for public IP networks", 5th IEEE International Conference on Industrial Informatics", pp. 189-194.

Sauter, T. "The three generations of field-level networks—Evolution and compatibility issues", IEEE Trans. Ind. Electron., vol. 57, no. 11, pp. 3585–3595, Nov. 2010.

Shahraeini, M., Javidi, M. H. and Ghazizadeh, M. S., 2011, "Comparison Between Communication Infrastructures of Centralized and Decentralized Wide Area Measurement Systems", IEEE Transactions On Smart Grid, Vol. 2, No. 1, pp. 206-211.

Wilson, P. R. and Johnstone, M. S., 1993, "Truly Real-Time Non-Copying Garbage Collection", In E. Moss, P. R. Wilson, B. Zom, editors, Proceedings of OOPSLA/ECOOP'93 Workshop on Garbage Collection in Object-Oriented Systems.

7. RESPONSABILIDADE AUTORAL

Os autores são os únicos responsáveis pelo conteúdo do material impresso incluídos no seu trabalho.

COMMON CODE CONTROLLER USED IN INDUSTRIAL DEVICES PROGRAMMING AND CONFIGURATION

Gilson Fonseca Peres Filho, gilsonfonseca@yahoo.com.br[1]
Marcio José da Cunha, cunhamjg@gmail.com[1]
Carlos Augusto Bissochi Junior, bissochi.jr@gmail.com[1]
Josué Silva de Morais, josuemorais@yahoo.com.br[1]
Fábio Vincenzi Romualdo da Silva, fabiovince@gmail.com[1]

[1] Federal University of Uberlândia – UFU, Electrical Engeneering Faculty- FEELT, Control and Automation Research Group – NPCA, Av. João Naves de Ávila, 2121 Bloco 3N,Uberlândia, MG, Brasil

Abstract. *There currently exist various communication protocols that are used in industrial networks. This diversity makes it difficult for interconnection between networks as well as the configuration of supervisory systems such as; PLCs (Programmable Logic Controllers), transmitters, HMI (Human-machine Interface), among other industrial actuators and control devices. It therefore becomes common practice that each manufacturer develops and makes available their own configurator/programming interface for PLCs, transmitters and HMIs. This situation thus makes technical training difficult in the sense of realizing maintenance and expansion of industrial processes. Therefore, the creation of free code computational resources that make it easier for manufacturers and researchers to construct the standardized configurators would provide the means to reduce manufacture's investment levels along with improved conditions for specialized training. Besides this, the specifications of some of the open standards are very expensive, extensive and difficult to implement. In relation to productivity there is something wrong as a development team in each company reads, interprets, tests the equipment and posteriorly institutions test and validate that equipment for approval. The current configurators and equipment only make available resources for configuring traditional controls of the PID (Proportional-Integral-Derivative Controller) type. This is another fact which motivates the creation of configurators and equipment that allow for the facilitated implementation of more efficient controls. This is due to the fact that improvements in control can represent visible profits in the fabrication process and depending upon the process a large reduction in environmental impacts. Through the consideration of the previously mentioned advantages, this study proposes the creation of functions with the objective of standardizing the program's configuration process of industrial devices. The functions are bound to a virtual machine designed for systems that perform with low computational effort. This set of functions whose code will be open at the end of the finished study, and thus will allow the manufacturers and researchers to create configuration interfaces and standardized programs with the additional advantage of the implementation of personalized control algorithms.*

Keywords*: standardization, open code, research, computational effort, specialized training*

RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.