Proposal of a hardware architecture for communication of Jason agents with FPGA's

Bruno Policarpo Toledo Freitas [1,2], **Carlos Eduardo Pantoja**[1], **José Viterbo**[2]

[1] Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)

[2]Universidade Federal Fluminense (UFF)

{bruno.freitas,pantoja}@cefet-rj.br, viterbo@ic.uff.br

Abstract. *Agents are autonomous software entities capable of reacting to its environment and pursuing its own goals. A group of agents interacting between themselves is a Multiagent System (MAS). MAS's can be used to control hardware devices on robotic or embedded systems. In these systems, a common challenge is ensuring real-time responses to external real-world events. In this context, this work proposes to replace microcontrollers commonly used by Jason's ARGO agents with dedicated hardware, turning the decoding and acting processes faster. Initial simulation results indicate that the decoding process can be reduced to 15 processor clock cycles.*

Resumo. *Agentes são entidades de software autônomas capazes de reagir ao ambiente e agir em prol de seus próprios objetivos. Um grupo de agentes interagindo entre si é um Sistema Multiagente (SMA). SMAs podem ser utilizados para controlar hardware em sistemas robóticos ou embarcados. Nesses sistemas, um desafio comum é como garantir respostas em tempo real a eventos externos ao ambiente em que são executados. Nesse contexto, este trabalho propõe substituir microcontroladores normalmente utilizados por agentes em Jason ARGO por um hardware dedicado, tornando o processo de decodificação e atuação mais rápido. Resultados iniciais de simulação indicam que é possível reduzir o processo de decodificação a 15 ciclos de relógio.*

1. Introdução

Agentes são entidades computacionais capazes de tomar decisões de forma autônoma, exibindo proatividade na buscas pelos seus objetivos próprios e capazes de reagir de acordo com alterações em seu ambiente (Bordini et al. 2007). Um grupo de agentes interagindo entre si é um Sistema Multiagente (SMA). Um dos principais modelos de desenvolvimento de agentes consiste no modelo *crenças*, *desejos* e *intenções* (Belief-Desire-Intention, ou BDI em inglês), baseado no raciocínio humano, em que os desejos são seus objetivos; as crenças representam o seu conhecimento acerca do mundo, e as intenções representam os diversos caminhos que um agente pode tomar para atingir seus desejos (Bratman et al. 1988). Baseada nesse modelo, Jason é uma das principais linguagens de programação para desenvolvimento de agentes, baseada no AgentSpeak (Rao 1996).

Por terem como característica principal a autonomia, os agentes têm sido cada vez mais utilizados em sistemas robóticos e embarcados. Outro tipo de solução bastante comum também tem sido integrar ao middleware ROS (Robotics 2026) a camada de

raciocínio em Jason (Gavigan and Esfandiari 2021; de Brito 2025; Davoust et al. 2020). Dentro de sistemas robóticos e embarcados, existem diversas situações em que a aplicação necessita de tomadas de decisão em tempo real como, por exemplo, evitar danos ao sistema no caso de um prazo não cumprido.

Em se tratando de utilização em sistemas de tempo real, porém, Jason apresenta diversas limitações. Ele não possui estruturas nativas para programação em tempo real, uma vez que é implementado em Java (Alzetta et al. 2020b). Diversos trabalhos apontam as limitações de desempenho que os agentes em Jason podem sofrer, como (Miller and Esfandiari 2022) os quais concluíram que o tempo de execução de um ciclo de raciocínio é polinomial em relação ao número de percepções e linear em relação ao número de planos e crenças. De maneira análoga, (Stabile and Sichman 2015) concluíram que o maior peso no desempenho do ciclo de raciocínio reside na atualização das percepções. Já (Gavigan and Esfandiari 2024) demonstraram que, dependendo da forma como um agente é programado, o peso no tempo de execução de um ciclo de raciocínio pode aumentar em até 4 vezes.

Em vista de tais limitações, diversas propostas visam aprimorar a reatividade dos agentes Jason. O trabalho de (Buss Becker et al. 2025) propõe uma camada reativa executada antes do ciclo de raciocínio, o qual avalia e responde a percepções críticas. Já (Traldi et al. 2022) adotam a estratégia de separar o processamento de tempo real de um ciclo de raciocínio, porém adotando PDDL como engine de agentes. Indo além, (Alzetta et al. 2020a) modificam a função de seleção de intenções do agente de forma a incorporar conceitos de tempo real, como utilização de carga computacional disponível. Todas essas soluções também utilizam simulações para análise e demonstração de desempenho, o que pode ocultar o risco do efeito da execução de outros processos do sistema operacional ao serem executadas em hardware real.

Este trabalho apresenta uma proposta de arquitetura de um hardware dedicado para melhorar o tempo de resposta de agentes Jason que atuam no mundo real. Diferentemente das soluções que atuam no software do agente, ela consiste em substituir microcontroladores por um hardware em FPGA, explorando a capacidade de paralelização dos mesmos para agilizar o processo de decodificação e resposta de agentes ARGO - um tipo de agente Jason que controla microcontroladores. Resultados iniciais indicam que é possível reduzir o tempo de decodificação de uma mensagem de getPercepts do ARGO para até 15 ciclos de relógio do FPGA.

Este trabalho está estruturado da seguinte maneira: a Seção 2 apresenta a Fundamentação Teórica. A Seção 3 apresenta a arquitetura proposta. A Seção 4 apresenta os resultados das simulações da arquitetura. A Seção 5 apresenta os trabalhos relacionados. Finalmente, a Seção 6 apresenta a conclusão.

2. Referencial teórico

2.1. Jason e agentes ARGO

Agentes são entidades de software autônomas, ditas proativas, pois são capazes de executar seus próprios objetivos, e reativas, pois são capazes de reagir a eventos externos. Agentes podem ser programados de diversas arquiteturas, dentre as quais o modelo BDI se destaca por ser baseado no próprio raciocínio humano. Dentro desse modelo, agentes possuem desejos, que representam o que desejam atingir; crenças, que representam

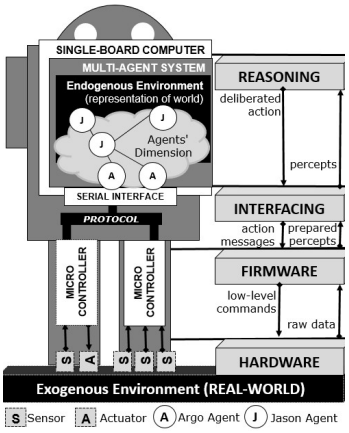


Figura 1. Arquitetura ARGO para desenvolvimento de SMA's robóticos e embarcados

informações sobre seu ambiente; e intenções, que são as diferentes estratégias que podem adotar para atingir um desejo.

Para programar um agente, é necessário um paradigma de programação, o qual chamamos de *Multiagent Oriented Programming* (MAOP) (Boissier et al. 2013). Dentre os MAOP's BDI existentes, um dos mais utilizados é Jason.

Agentes também podem ser utilizados no desenvolvimento de sistemas robóticos ou embarcados. Para isso, é necessária alguma interface para representação de informações vindas do hardware em forma de crenças na mente de um agente Jason. Para isso, uma possível solução é adotar a arquitetura ARGO (Pantoja et al. 2016), conforme mostrado na Figura 1. Nela, é adotada uma arquitetura em quatro camadas: hardware, que consiste na implementação física no mundo real; firmware, que consiste na programação do software do microcontrolador; interface, que consiste no protocolo de comunicação entre o microcontrolador e o sistema operacional; e a camada de raciocínio, que no ARGO é desenvolvida usando Jason.

Para realizar a comunicação com o microcontrolador, o agente ARGO implementa novas ações internas, as quais são:

- **.argo.port(Port)** : define a porta de comunicação serial entre o computador e o microcontrolador
- **.argo.percepts.open()**: abre o canal de comunicação entre o agente e o microcontrolador para envio de ações e atualização de crenças
- **.argo.limit(Tempo)**: Define o período de atualização das percepções
- **.argo.act(Action)**: Envia ao microcontrolador uma ação *Action*. Tal ação deve estar previamente definida no firmware do mesmo

Após a abertura da porta serial com o open e a definição do período com o limit, o agente ARGO atualiza periodicamente, em forma de crenças, os dados oriundos do microcontrolador. Esse processo é transparente e é chamado de *getPercepts* dentro da arquitetura.

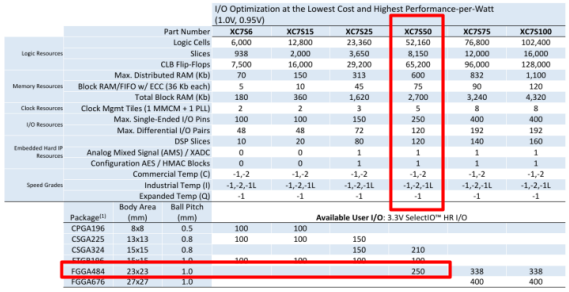
2.2. FPGA

Field Programmable Gate Array (FPGA) é uma tecnologia que permite a criação de hardware digital customizado. Para desenvolver FPGAs, são utilizadas linguagens de

descrição de hardware, que descrevem o comportamento de circuitos eletrônicos, tal como VHDL (Pedroni 2010).

FPGA's possuem famílias, onde cada família possui determinadas características que variam de acordo com o fabricante (Xilinx 2026). Porém, em geral, as mais importantes costumam ser a quantidade de Flip-Flops (FF), usados para criar circuitos sequenciais; Lookup Tables (LUT), utilizadas para criar circuitos combinacionais; e a quantidade de portas de entrada e de saída, que limitam o número de "fios"que um circuito em FPGA pode ter com o mundo exterior. Tais parâmetros são fatores limitantes no desenvolvimento de hardware, determinando a complexidade máxima que ele pode alcançar.

Como exemplo desses conceitos, considere a família XC7S50FGGA484-1, utilizada neste trabalho e mostrada na Figura 2. Ela pertence à família Spartan7 (Xilinx 2026) da Xilinx, possuindo 52160 portas lógicas e 65200 flip-flops, ambas as informações destacadas em vermelho.



I/O Optimization at the Lowest Cost and Highest Performance-per-Watt (1.0V, 0.95V)

	Part Number	XC7S6	XC7S15	XC7S25	XC7S50	XC7S75	XC7S100
Logic Resources	Logic Cells	6,000	12,800	23,360	52,160	76,800	102,400
	Slices	938	2,000	3,650	8,150	12,000	16,000
	CLB Flip-Flops	7,500	16,000	29,200	65,200	96,000	128,000
	Max. Distributed RAM (Kb)	70	150	313	600	832	1,100
Memory Resources	Block RAM/FIFO w/ ECC (36 Kb each)	5	10	45	75	90	120
	Total Block RAM (Kb)	180	360	1,620	2,700	3,240	4,320
Clock Resources	Clock Mgmt Tiles (1 MMCM + 1 PLL)	2	2	3	5	8	8
I/O Resources	Max. Single-Ended I/O Pins	100	100	150	250	400	400
	Max. Differential I/O Pairs	48	48	72	120	192	192
Embedded Hard IP Resources	DSP Slices	10	20	80	120	140	160
	Analog Mixed Signal (AMS) / XADC	0	0	1	1	1	1
	Configuration AES / HMAC Blocks	0	0	1	1	1	1
Speed Grades	Commercial Temp (C)	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2	-1,-2
	Industrial Temp (I)	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L	-1,-2,-1L
	Expanded Temp (Q)	-1	-1	-1	-1	-1	-1

Package[1]	Body Area (mm)	Ball Pitch (mm)	Available User I/O: 3.3V SelectIO™ HR I/O					
CPGA196	8x8	0.5	100	100				
CSGA225	13x13	0.8	100	100	150			
CSGA324	15x15	0.8			150	210		
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]		
FGGA484	23x23	1.0				250	338	338
FGGA676	27x27	1.0					400	400

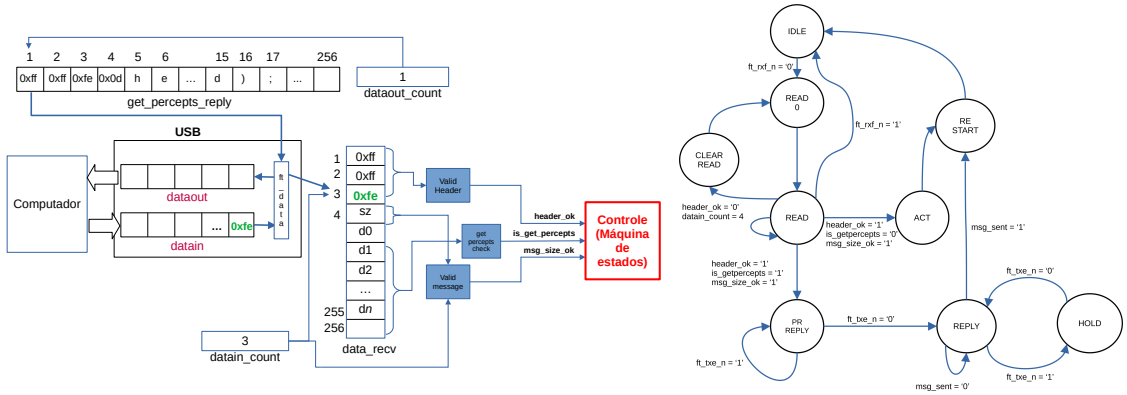
Figura 2. Características da família de FPGAs Spartan7 da Xilinx

3. Proposta

Com o objetivo de melhorar o tempo de resposta de um agente controlando um sistema robótico, este trabalho propõe uma arquitetura de hardware para comunicação com agentes ARGO. Com base no protocolo Javino utilizado pelo agente ARGO, a arquitetura explora uma característica inerente a um FPGA: a capacidade de paralelização. Assim, substitui-se o processo sequencial de decodificação de mensagens pelo processador do microcontrolador por um processo paralelo no FPGA. O mesmo ocorre também no processo de codificação das percepções para o envio de mensagens do FPGA ao agente ARGO. Além disso, remove-se o uso da memória, eliminando a latência associada ao seu uso.

A Figura 3a apresenta a visão geral da arquitetura. A porta USB possui dois buffers FIFO *datain* e *dataout* que recebem bytes oriundos do computador para serem lidos pelo FPGA, e vice-versa. O bloco *ft_data* representa um barramento de dados por onde flui o fluxo de dados anterior. Quando um dado é recebido pela interface USB, ele é salvo na posição *datain_count* do vetor de bytes *data_recv* e, em seguida, é incrementado. Na Figura, o byte 0xfe recebido via USB seria salvo na posição 3 de data_recv. Os blocos em azul são circuitos combinacionais que verificam continuamente o conteúdo de "data_recv". O bloco "*Valid Header*"verifica se o que está salvo nas posições 1 a 3 corresponde a um cabeçalho Javino válido: se, após ler 3 bytes, o conteúdo não for "0xff 0xff 0xfe", ele reinicia o processo de leitura (datain_count = 1). O bloco "*Valid message*"verifica se a mensagem esperada já foi completamente recebida, o que é feito ao verificar se "datain_count - 4"é igual ao byte 4 (sz), o qual consiste no tamanho da

mensagem recebida. Já o bloco ”*get_percepts_check*”verifica se a mensagem é ou não um getPercepts. Se o cabeçalho é válido; o tamanho total esperado da mensagem já foi recebida; e a mensagem é um getPercepts, é iniciado o processo de resposta. Caso contrário, a mensagem é interpretada como uma ação.



(a) Visão geral **(b) Máquina de estados**

Figura 3. Arquitetura do hardware

A Figura 3b apresenta a máquina de estados do funcionamento da arquitetura. Os sinais prefixados com ”ft”representam os sinais de controle do hardware USB. Esses estados são:

IDLE: estado padrão que representa o hardware ocioso.

READ_0: estado que representa a primeira fase do início da recepção, em que o barramento de comunicação *ft_data* é colocado no modo de recepção.

READ: nesse estado, a cada ciclo de clock, os dados recebidos são colocados em um buffer onde, de forma paralela, é feita toda a decodificação necessária para recepção de uma mensagem.

CLEAR_READ: Caso o hardware não detecte cabeçalho válido recebido, ele passa a este estado, onde o que foi recebido é descartado antes de retornar ao estado READ_0

PR_REPLY: Caso o hardware detecte uma mensagem de getPercepts recebida, ele passa para este estado intermediário, que prepara o processo de resposta.

REPLY: Realiza uma resposta a um getPercepts. Neste trabalho, é enviada como resposta a percepção *hello(world);* do FPGA para um agente ARGO.

HOLD: Caso o buffer de envio esteja cheio (sinal $ft\_txe\_n =' 1'$), o hardware é posto neste estado de espera até poder ser liberado a continuar

RESTART: A mensagem de resposta ou act foi enviada, restaurando o hardware para aguardar a recepção de uma nova mensagem.

ACT: Representa uma ação enviada por um agente ARGO.

4. Resultados

A Figura 4 ilustra a execução da arquitetura no simulador integrado da ferramenta Vivado. Os pontos importantes da simulação estão destacados de azul. No instante de

tempo 110ns, o processo de recebimento de dados é iniciado, com *ft_oe_n* igual a 0 para colocar o barramento em modo de recebimento e *ft_rd_n* igual a 0 para indicar o início do processo. No instante de tempo 190ns, é decodificado o preâmbulo *ffe*, marcando que a mensagem recebida é uma mensagem válida, e verificado que a mensagem recebida possui 11 caracteres . Entre 190ns e 410ns são recebidos os 11 caracteres que compõem a palavra *getPercepts*. No instante 410ns, é concluída a decodificação, com *header_ok*, *is_get_percepts*, e *msg_size_ok* iguais a '1'. Finalmente, no instante de tempo 410ns, é iniciado o processo de resposta do getPercepts, colocando *ft_wr_n* igual a '0'.

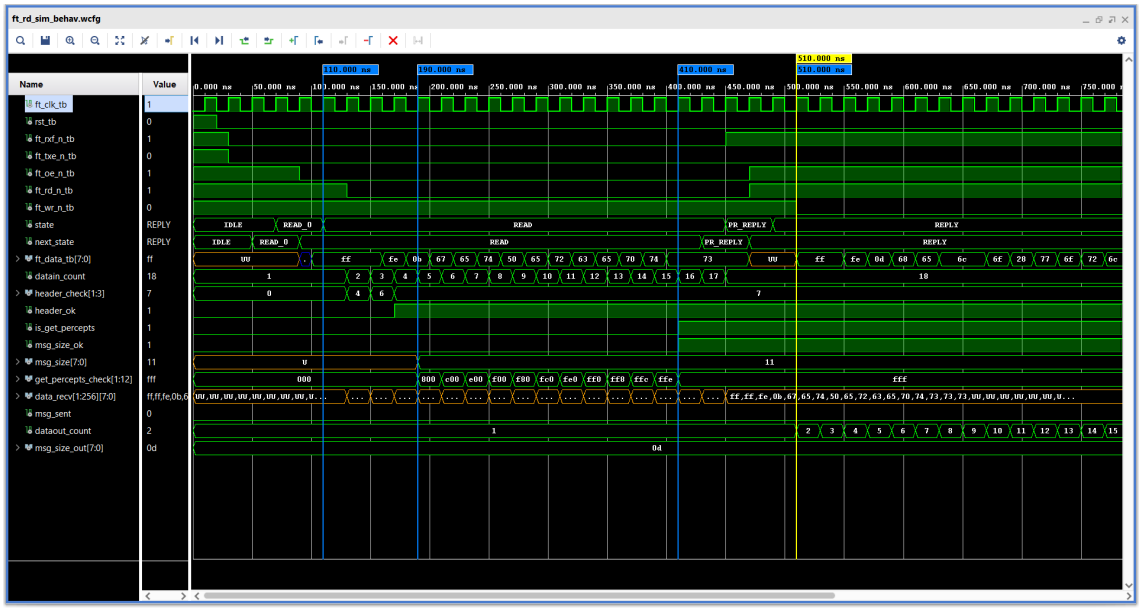


Figura 4. Resultados da simulação no Vivado

4.1. Reprodutibilidade

É disponibilizada uma máquina virtual para a reprodutibilidade das simulações[1] . Para executar a simulação a partir do código-fonte, é necessário instalar a ferramenta Vivado da AMD[2]

5. Trabalhos relacionados

Diversos trabalhos propõem melhorar o tempo de resposta de agentes cognitivos. Porém, grande parte deles propõe alterações no framework BDI, desconsiderando o efeito da execução de outros processos pelo sistema operacional. O trabalho de (Alzetta et al. 2020a) propõe adicionar ao modelo BDI os principais conceitos de sistemas de tempo real: prazos e prioridades. Dentro do modelo proposto, é alterada a função de seleção de planos, de forma a selecionar um plano que maximize uma função de utilidade, a qual leva em consideração a prioridade; o tempo de computação; e a banda computacional disponível para realização de uma tarefa.

[1] https://papers.chon.group/WESAAC/2026/hardwareJasonFPGA/

[2] https://www.amd.com/pt/support/downloads/adaptive-socs-and-fpgas/development-tools/2025-2.html

Indo além, (Calvaresi et al. 2021) propõem um modelo formal de um SMA de tempo real, criando um protocolo de negociação de tarefas denominado Reservation-Based Negotiation. Nele, a negociação de tarefas em tempo real entre os agentes de um MAS leva em consideração os tempos de computação e seus prazos, de modo que as tarefas de negociação recebidas por um agente só são aceitas se ele puder escaloná-las. Os experimentos realizados pelo trabalho envolvem a utilização de simuladores, sendo ainda a discutir sua aplicação em SMAs robóticos ou como integrar com SOs de tempo real.

Outra estratégia utilizada é separar do modelo BDI o monitoramento e a infraestrutura de execução de tarefas de tempo real, adotado por (Traldi et al. 2022). Durante um ciclo de raciocínio, após avaliar quais intenções estão ativas, é ativada a camada de tempo real, a qual verifica se uma dada tarefa pode ser realizada de acordo com suas restrições, dentre as quais a sua escalonabilidade. Caso possível, a tarefa do plano é enviada para a camada de execução de tarefas de tempo real. Dentre os trabalhos aqui apresentados, este separa do mecanismo BDI a análise de eventos de tempo real. Porém, ele utiliza PDDL e também utiliza simuladores, muito embora o trabalho cite (sem exemplificar) que é possível aplicar a solução em sistemas operacionais de tempo real.

Já (Buss Becker et al. 2025) adicionam, antes do ciclo de raciocínio principal do Jason, uma camada reativa capaz de analisar ditas percepções críticas, reagindo prioritariamente a elas. Dentro da proposta, as percepções críticas são indicadas por meio de anotações dentro do código Jason e as reações necessariamente são únicas e avaliadas antes do ciclo BDI. Ao alterar apenas o ciclo, tal tipo de solução ainda é vulnerável a oscilações de carga computacional do sistema operacional residente.

6. Conclusão

Este trabalho apresentou uma proposta de uma arquitetura de hardware para atender a requisitos de resposta a eventos em tempo real. A arquitetura busca agilizar o processo de decodificação e ação de agentes ARGO, explorando o paralelismo intrínseco proporcionado por FPGAs. Como trabalhos futuros, é importante generalizar a tanto a composição de respostas do getPercepts quanto os modelos de ação do ARGO.

Referências

Alzetta, F., Giorgini, P., Marinoni, M., and Calvaresi, D. (2020a). RT-BDI: A Real-Time BDI Model. In *Lect. Notes Comput. Sci.*, volume 12092 LNAI, pages 16–29. Springer. Journal Abbreviation: Lect. Notes Comput. Sci.

Alzetta, F., Giorgini, P., Najjar, A., Schumacher, M. I., and Calvaresi, D. (2020b). In-Time Explainability in Multi-Agent Systems: Challenges, Opportunities, and Roadmap. In Calvaresi, D., Najjar, A., Winikoff, M., and Främling, K., editors, *Explainable, Transparent Autonomous Agents and Multi-Agent Systems*, pages 39–53, Cham. Springer International Publishing.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with JaCaMo. *Science of Computer Programming*, 78(6):747–761.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming multi-agent systems in AgentSpeak using Jason*. Wiley series in agent technology. Wiley, Chichester.

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(3):349–355.

Buss Becker, L., de Oliveira Silvestre, I., Hübner, J. F., and Fisher, M. (2025). An expedited BDI agent architecture: Improving the responsiveness of agent-based autonomous systems for handling critical situations. *Robotics and Autonomous Systems*, 186:104917.

Calvaresi, D., Dicente Cid, Y., Marinoni, M., Dragoni, A. F., Najjar, A., and Schumacher, M. (2021). Real-time multi-agent systems: rationality, formal model, and empirical results. *Autonomous Agents and Multi-Agent Systems*, 35(1):12.

Davoust, A., Gavigan, P., Ruiz-Martin, C., Trabes, G., Esfandiari, B., Wainer, G., and James, J. (2020). An architecture for integrating BDI agents with a simulation environment. In *Lect. Notes Comput. Sci.*, volume 12058 LNAI, pages 67–84. Springer. Journal Abbreviation: Lect. Notes Comput. Sci.

de Brito, M. (2025). ROS-BDI Robots: An Agent-Based Approach for Programming the Behaviour of Autonomous Robots. *J. Emerg. Technol. Comput. Syst.*, 22(1):10:1–10:26.

Gavigan, P. and Esfandiari, B. (2021). Agent in a Box: A Framework for Autonomous Mobile Robots with Beliefs, Desires, and Intentions. *Electronics*, 10(17):2136.

Gavigan, P. and Esfandiari, B. (2024). Quantifying the relationship between software design principles and performance in Jason: a case study with simulated mobile robots. *Annals of Mathematics and Artificial Intelligence*, 92(4):775–795.

Miller, J. and Esfandiari, B. (2022). Analysis of the Execution Time of the Jason BDI Reasoning Cycle. In Alechina, N., Baldoni, M., and Logan, B., editors, *Engineering Multi-Agent Systems*, pages 218–236, Cham. Springer International Publishing.

Pantoja, C. E., Stabile, M. F., Lazarin, N. M., and Sichman, J. S. (2016). ARGO: An Extended Jason Architecture that Facilitates Embedded Robotic Agents Programming. In Baldoni, M., Müller, J. P., Nunes, I., and Zalila-Wenkstern, R., editors, *Engineering Multi-Agent Systems*, Lecture Notes in Computer Science, pages 136–155, Cham. Springer International Publishing.

Pedroni, V. A. (2010). *Circuit design and simulation with VHDL*. MIT Press, Cambridge, Mass, 2nd ed edition.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In Van de Velde, W. and Perram, J. W., editors, *Agents Breaking Away*, pages 42–55, Berlin, Heidelberg. Springer.

Robotics, O. (2026). ROS: Home. Disponível em: `www.ros.org` . Acesso em: 14 / 8 / 2026.

Stabile, M. F. and Sichman, J. S. (2015). Evaluating Perception Filters in BDI Jason Agents. In *2015 Brazilian Conference on Intelligent Systems (BRACIS)*, pages 116–121.

Traldi, A., Bruschetti, F., Robol, M., Roveri, M., and Giorgini, P. (2022). Real-Time BDI Agents: A Model and Its Implementation. volume 1, pages 511–517.

Xilinx (2026). Spartan 7 family overview. Disponível em: `https://docs.amd.com/v/u/en-US/7-series-product-selection-guide`. Acesso em : 11 / 6 / 2026.