5
Exemplos de circuitos

Nesta seção, descrevemos alguns exemplos que provém uma discussão mais detalhada acerca da mecânica de circuitos de objetos. De início são mostrados dois circuitos muito simples, capazes de computar funções matemáticas elementares, e a seguir a discussão continua com aplicações mais concretas.

5.1.Função fatorial

Neste exemplo, construiremos um circuito de objetos capaz de computar, iterativamente, a função fatorial, à medida que sinais de relógio são disparados. O primeiro diagrama da Ilustração 1 mostra o circuito completo. Ele contém dois CIOs: um somador, rotulado SUM; e um multiplicador, denominado MULT. O pequeno círculo com o símbolo de adição denota um dispositivo fonte, que provê um valor de potencial constante em sua saída - neste exemplo, um objeto representando o valor inteiro 1.

Por simplicidade, ambos os CIOs têm, cada qual, um único barramento, com portas de nome X0, X1, CK e C. As portas X0 e X1 são de entrada; elas provém os argumentos para as respectivas funções implementadas pelos CIOs. CK denota uma porta por onde são recebidos sinais de relógio. Neste exemplo, um sinal de relógio causa a leitura das portas de entrada pelo respectivo CIO, que então recalcula sua função e escreve o novo valor na porta de saída. Finalmente, a porta C é usada para reinicialização do dispositivo: um sinal nesta porta leva o CIO ao seu estado inicial.

As duas portas Y são as saídas do somador e multiplicador. Em qualquer instante após a primeira iteração, seus potenciais correspondem ao valor computado após a chegada do último sinal de relógio.

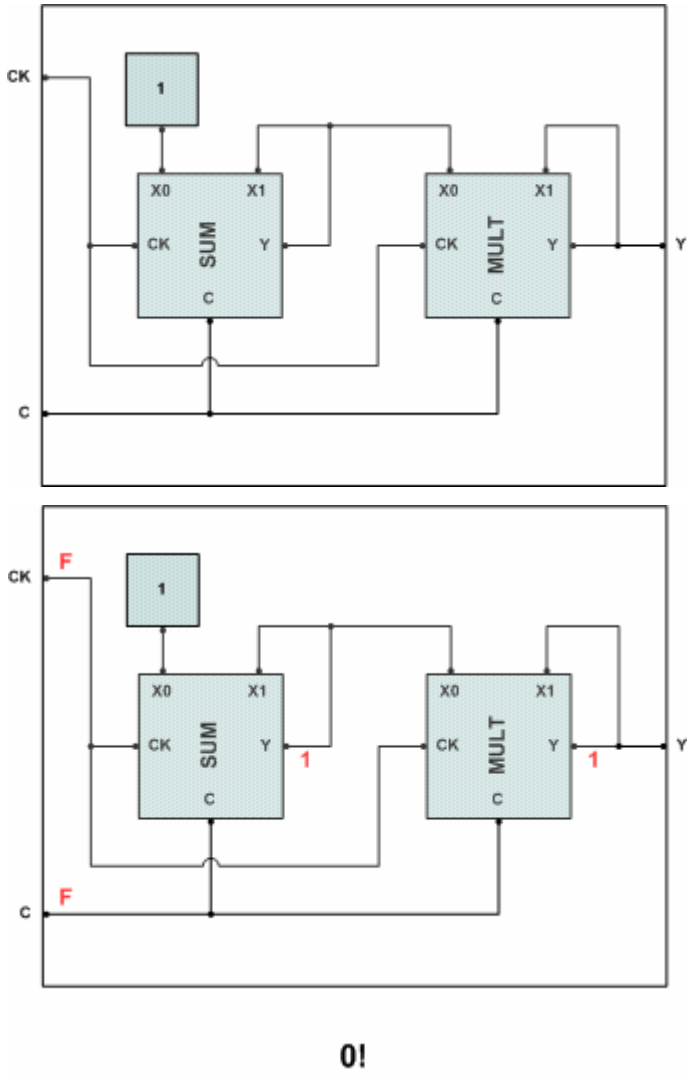
O segundo diagrama na Ilustração 1 mostra o estado inicial do circuito. Supõe-se que um circuito externo é o responsável pela geração de sinais de relógio e de reinicialização, representados por objetos booleanos. Os potenciais iniciais nas portas de saída Y não são resultado de nenhuma computação. Eles

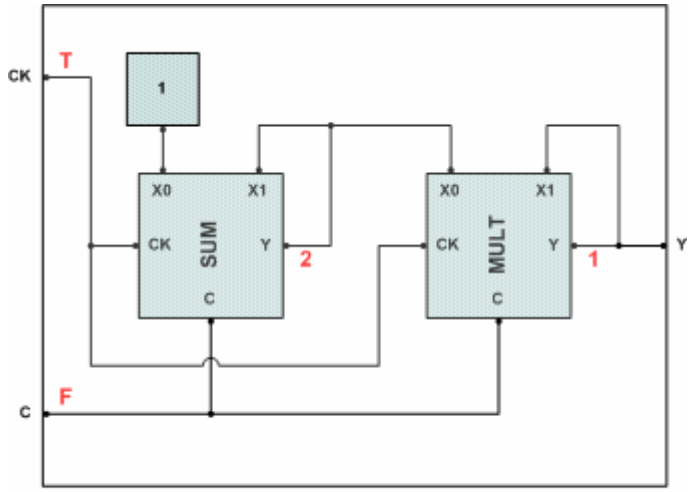
foram pré-programados no dispositivo, sendo automaticamente reescritos sempre que ele é reinicializado. Esta abordagem permite que seja modelado o estado transiente de um sistema.

A seguir, nota-se a chegada de um sinal de relógio, que é propagado até as portas CK de ambos os CIOs, fazendo com que suas funções internas sejam imediatamente recalculadas. O somador lê em X0 e X1 o valor 1, dando 2 como resultado. O multiplicador, por sua vez, também avalia X0 e X1 como 1, calculando 1. Na prática, este passo corresponde ao cálculo de 1!, que pode ser lido na porta F.

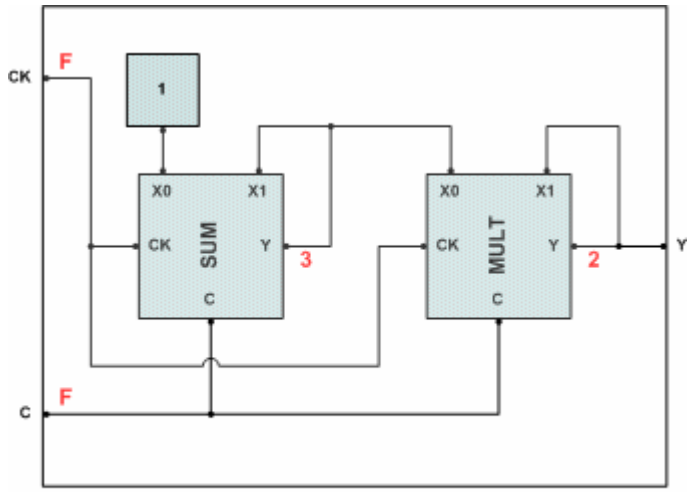
É fácil ver que a i-ésima transição de relógio corresponde à computação de i!. Os próximos dois diagramas mostram o cálculo de 2! e 3!, respectivamente. Finalmente, o ultimo diagrama ilustra um sinal de reinicialização, que traz o circuito de volta ao seu estado inicial.

Ilustração 1 O circuito de fatorial e alguns estágios da computação. O último estágio é a chegada de um sinal de reinicialização, trazendo o circuito ao seu estado inicial.

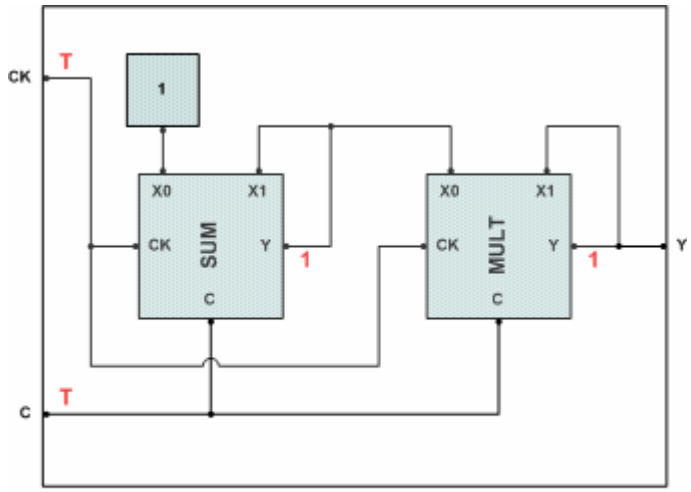




1!



2!



0!

O circuito de fatorial foi implementado utilizando o JOCA. A Ilustração 2 mostra a tela de saída produzida ao executar o circuito. Cada linha contém três informações:

- O dispositivo sendo executado: MUL (multiplicador) ou SUM (somador).
- Os valores atuais nas portas de entrada X0 e X1.
- O resultado da computação do dispositivo, escrito na porta de saída Y.

Note que o valor da função fatorial em si é, a cada iteração, o próprio resultado da computação do dispositivo MULT. A seqüência segue até o cálculo do fatorial de 17.

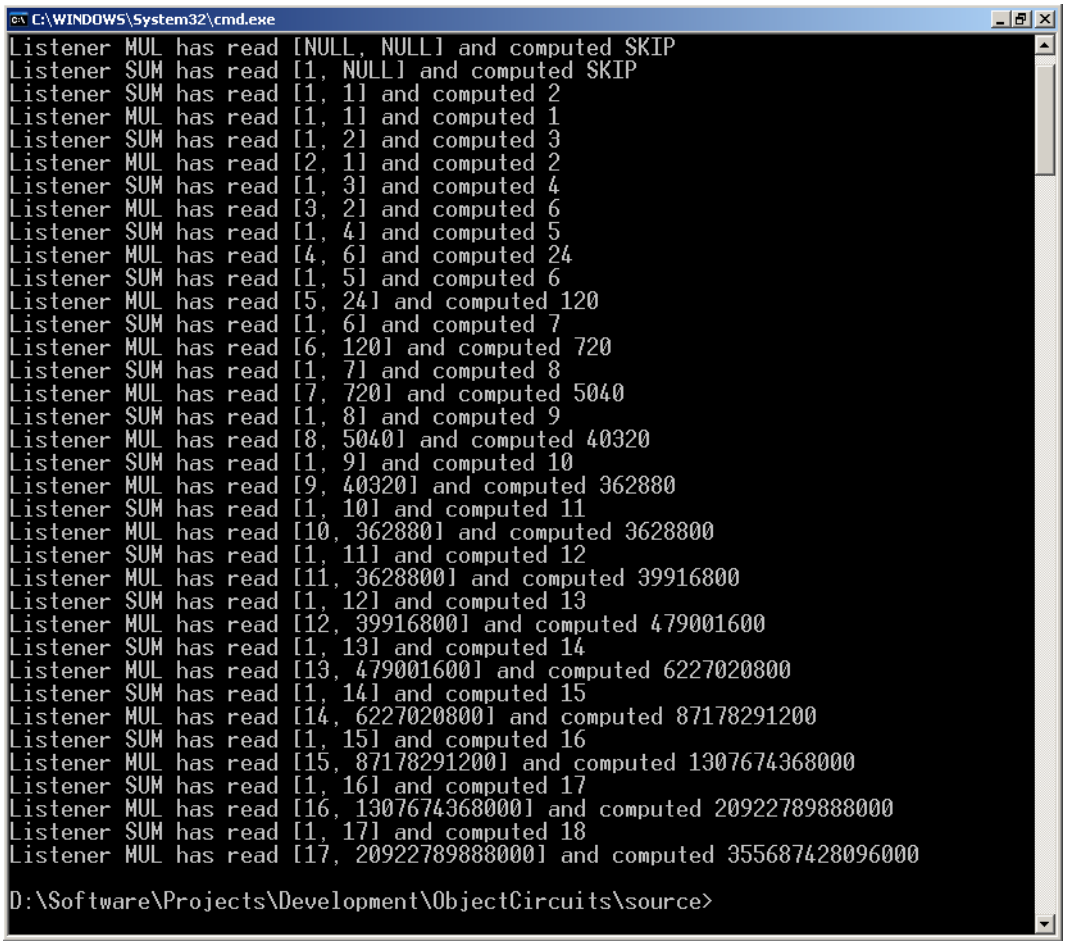


Ilustração 2 Resultados produzidos por uma implementação do circuito de fatorial utilizando a arquitetura JOCA.

5.1.1.Temporização

Uma análise cuidadosa do circuito de fatorial suscitará um problema interessante: como garantir que, entre um sinal e outro de relógio, haverá tempo hábil para que cada dispositivo realize sua computação? Caso não haja, os dispositivos ficarão fora de sincronia e o sistema como um todo entrará num estado espúrio.

Algumas soluções são possíveis. Se o ciclo de cada dispositivo for conhecido, basta ajustarmos a freqüência do relógio para o valor apropriado. Se, por outro lado, o ciclo não for conhecido, ou existirem dispositivos cujo ciclo é variável, a alternativa mais simples é a implementação de um relógio "inteligente", que perceba o término da computação por cada um dos dispositivos envolvidos, e só então emita um novo sinal. A lógica deste relógio é simples e deixamos como exercício ao leitor.

Uma abordagem mais robusta e completa seria a criação de mecanismos que permitam especificar o comportamento do dispositivo ao longo do tempo, de modo que as restrições temporais sejam conhecidas e passem a ser uma preocupação do designer do circuito. Tais mecanismos ainda não existem no modelo atual de circuitos de objetos.

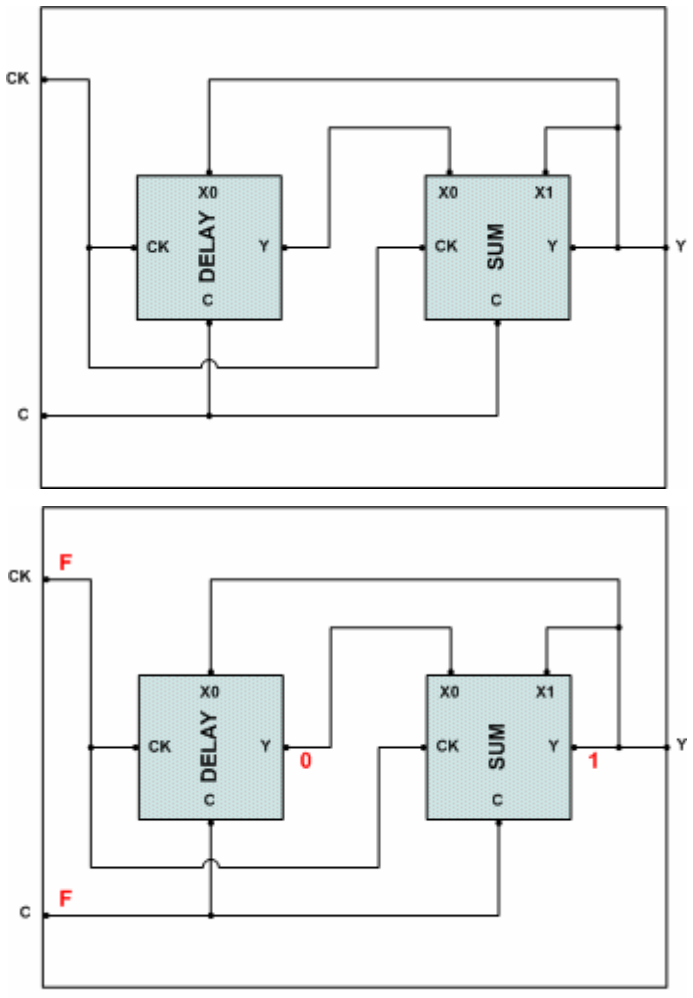
5.2.Função de Fibonacci

Descreveremos agora a implementação da função de Fibonacci, $f(n) = f(n-1) + f(n-2)$, com $f(0) = f(1) = 1$. Este exemplo é, em muitos aspectos, similar ao anterior, mas ilustra o uso de um novo dispositivo: o *retardo* ou *delay*. A Ilustração 3 mostra o diagrama completo do circuito e alguns estágios da computação.

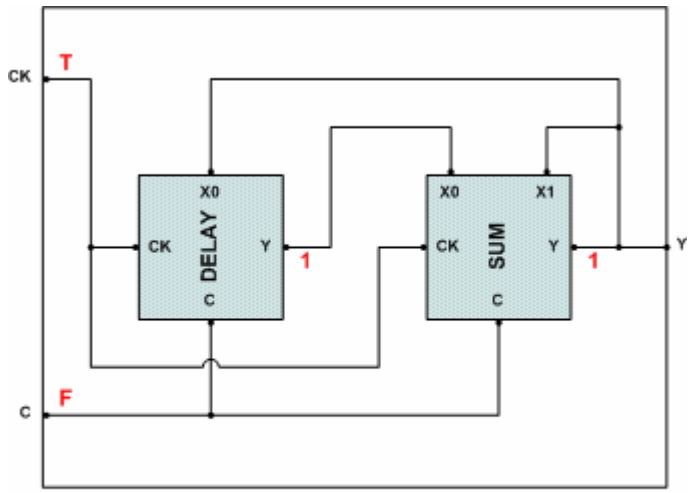
Na função de Fibonacci, o valor calculado para $f(k)$ num dado passo será usado ainda nos dois passos seguintes, onde tal valor assumirá, respectivamente, o papel de $f(k-1)$ e $f(k-2)$. Nossa implementação realiza a primeira "transformação", de $f(k)$ em $f(k-1)$, conectando a saída Y do dispositivo somador à sua entrada X1. Já a segunda transformação, de $f(k)$ em $f(k-2)$, dá-se pela conexão de Y a um dispositivo de retardo, que por sua vez é conectado à porta de entrada X0. Um exame cuidadoso mostrará que o potencial em X1 é igual ao detectado em Y na

transição de relógio anterior, ao passo que o potencial em X0 é igual ao que estava em Y duas transições atrás.

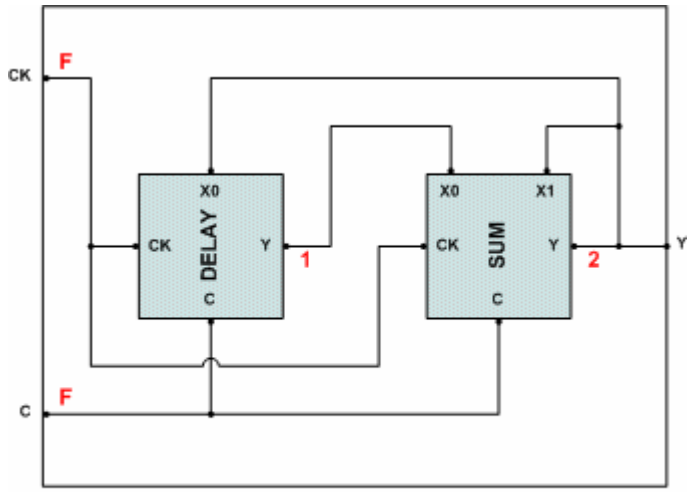
Ilustração 3 O circuito de Fibonacci e alguns estágios da computação.



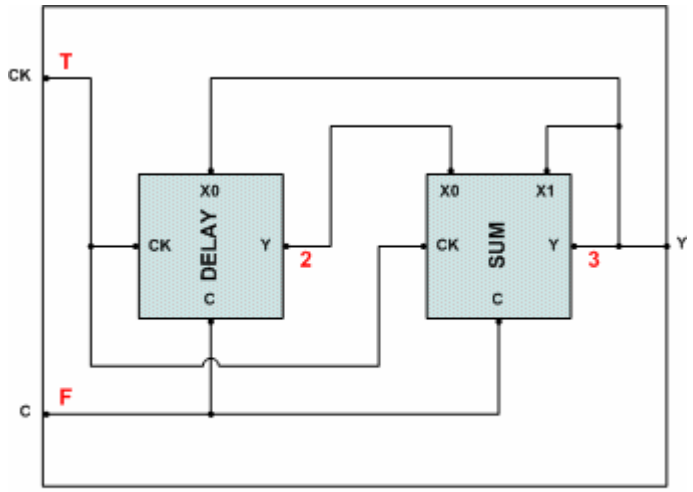
FIB(0)



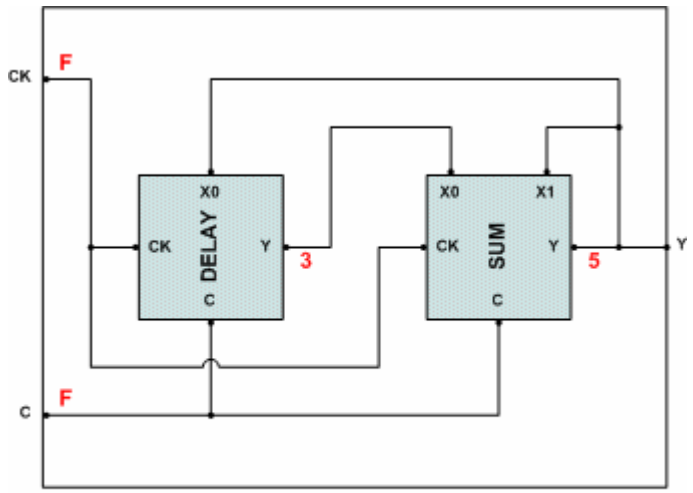
FIB(1)



FIB(2)



FIB(3)



FIB(4)

5.3.Jogo da vida

O Jogo da Vida (Berlekamp, 1982) foi criado pelo matemático John Conway em 1970, e logo se tornou um exemplo clássico de autômatos celulares. O jogo é praticado dispondo-se um número de células num reticulado bidimensional. Uma célula pode estar viva ou morta, e a cada passo, seu estado é recomputado segundo um conjunto de regras que leva em consideração o estado corrente da célula, bem como o das células vizinhas.

1. Uma breve descrição das regras é dada a seguir:
2. Uma célula com menos de duas ou mais de três vizinhas vivas torna-se ou permanece morta.
3. Uma célula com duas vizinhas vivas mantém seu estado corrente.
4. Uma célula com três vizinhas vivas torna-se ou permanece viva.

Implementamos uma célula como mostrado na Ilustração 4. Existem oito portas de entrada, às quais as células vizinhas se conectam. O circuito funciona da seguinte forma: ao sinal do relógio, o circuito SUM avalia o número de células vizinhas cujo estado é viva. Este valor, juntamente com o estado corrente da célula, é passado ao dispositivo F, que então computa o novo estado. O pseudocódigo da função implementada por F é dado a seguir:

```
1.  function Fn+1( Fn, L )
2.  {
3.     if ( L <= 1 ) or ( L >= 4 )
4.     {
5.        output 0
6.     }
7.
8.     if ( L = 2 )
9.     {
10.       output Fn
11.    }
12.
13.    output 1
14. }
```

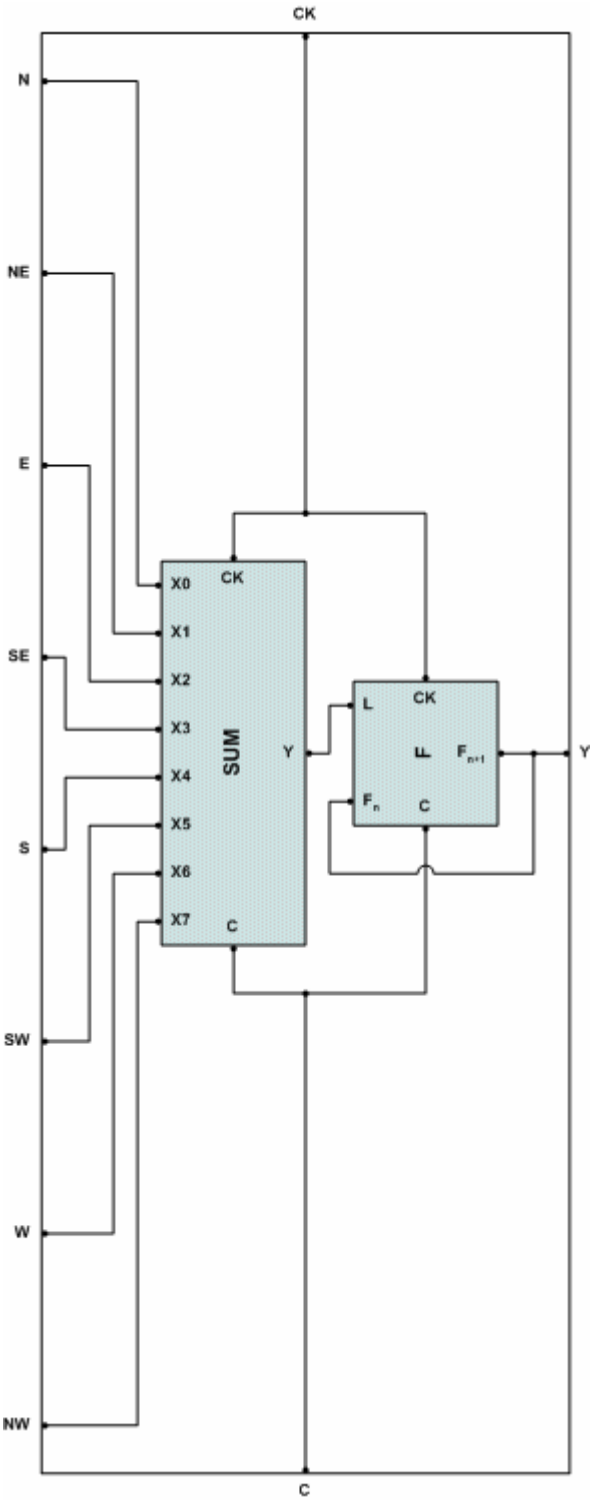


Ilustração 4 Visão interna de um dispositivo representando uma célula do Jogo da Vida. O dispositivo SUM calcula o número de células vizinhas vivas, enquanto F calcula o próximo estado da célula.

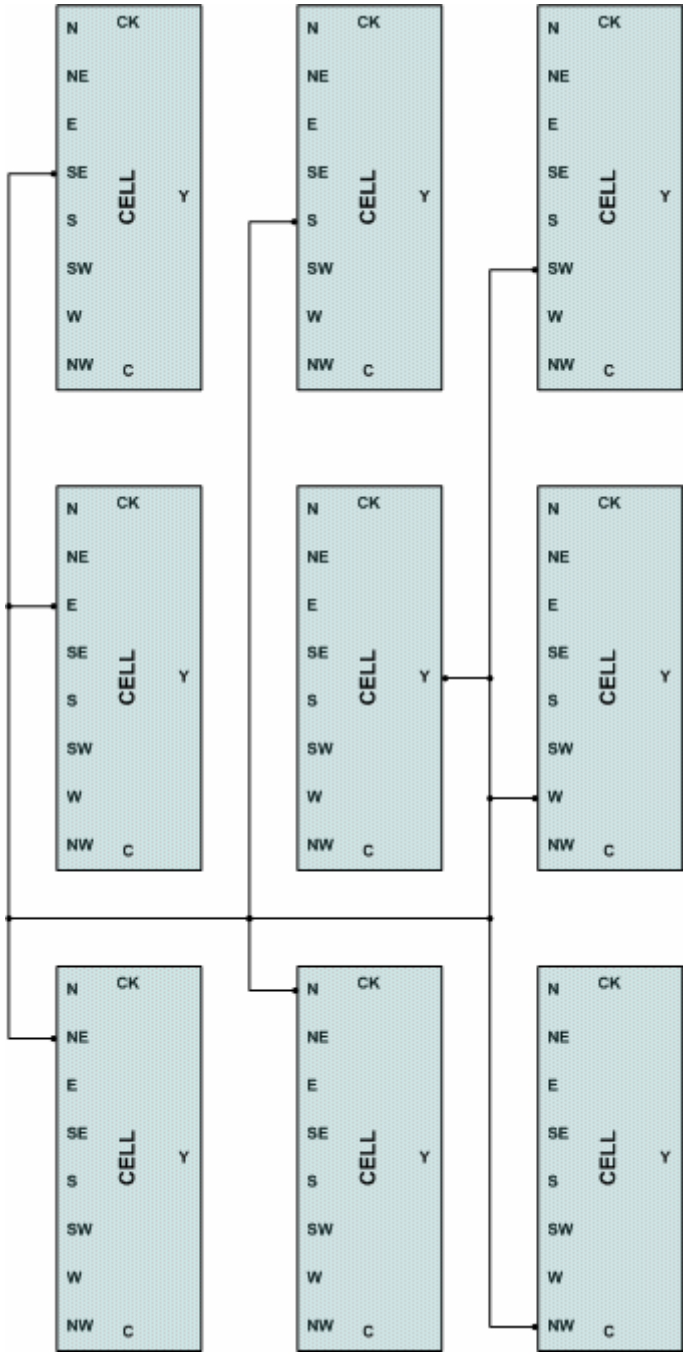


Ilustração 5 Reticulado de células 3x3. As linhas de transmissão mostradas conectam a porta de saída Y da célula central à porta correspondente de cada célula vizinha. Por simplicidade, as outras conexões foram omitidas.

5.4.Conversor de temperatura

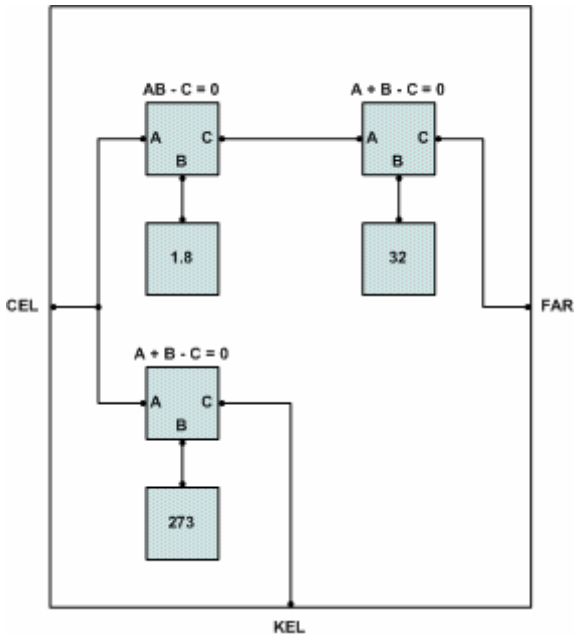
A bidirecionalidade na comunicação presente nos dispositivos permite a expressão de determinadas computações com mais elegância. Em particular, no

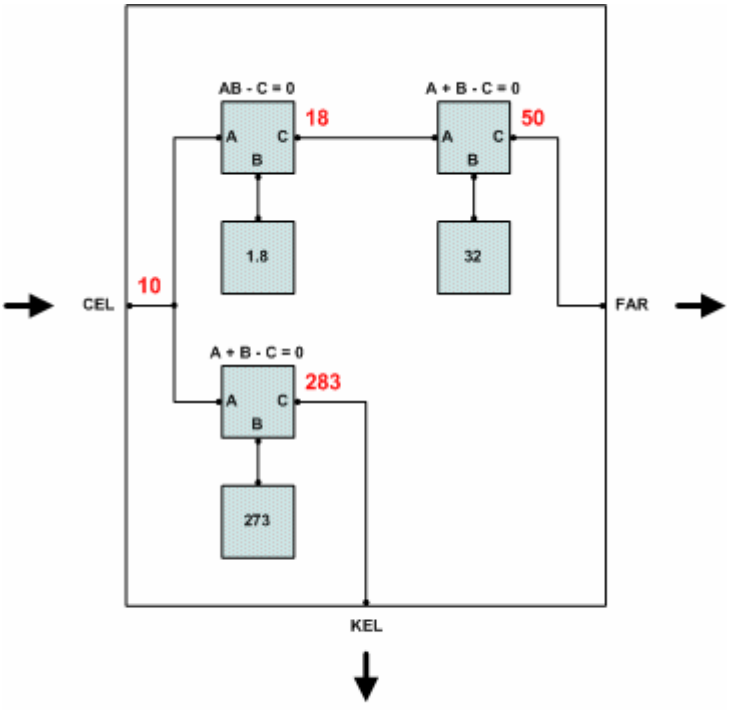
caso de funções bijetoras, a implementação é trivial e imediata. Considere o problema de converter o valor de uma temperatura de uma escala para outra. Uma implementação possível é a de um dispositivo com um barramento contendo duas portas: ao receber um valor em uma das portas, este é convertido e o resultado é escrito na outra. Não há uma porta de "entrada" e outra de "saída"; ambas funcionam nos dois sentidos, conforme a situação. Obviamente, somente uma das portas pode receber valores de cada vez, o contrário resultaria num curto-circuito em uma das portas.

A Ilustração 6 mostra um circuito conversor de temperatura envolvendo as escalas Celsius, Kelvin e Fahrenheit. Ele possui dois tipos de dispositivos: os conversores propriamente ditos (com portas A, B e C) e os que representam constantes numéricas. Cada dispositivo conversor funciona da seguinte forma: sempre que duas de suas portas possuírem valores de entrada, uma computação é feita e o resultado colocado na terceira porta.

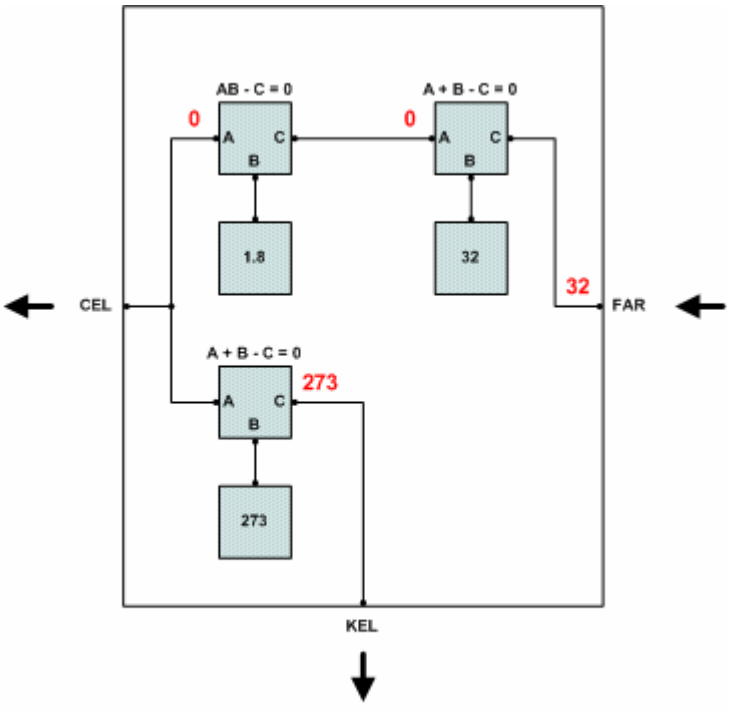
Ao aplicar um potencial em dos nós CEL, KEL ou FAR, este potencial é propagado internamente pelos dispositivos conversores de modo que os outros dois nós tem seus potencias preenchidos com o valor convertido.

Ilustração 6 O circuito conversor e algumas conversões.

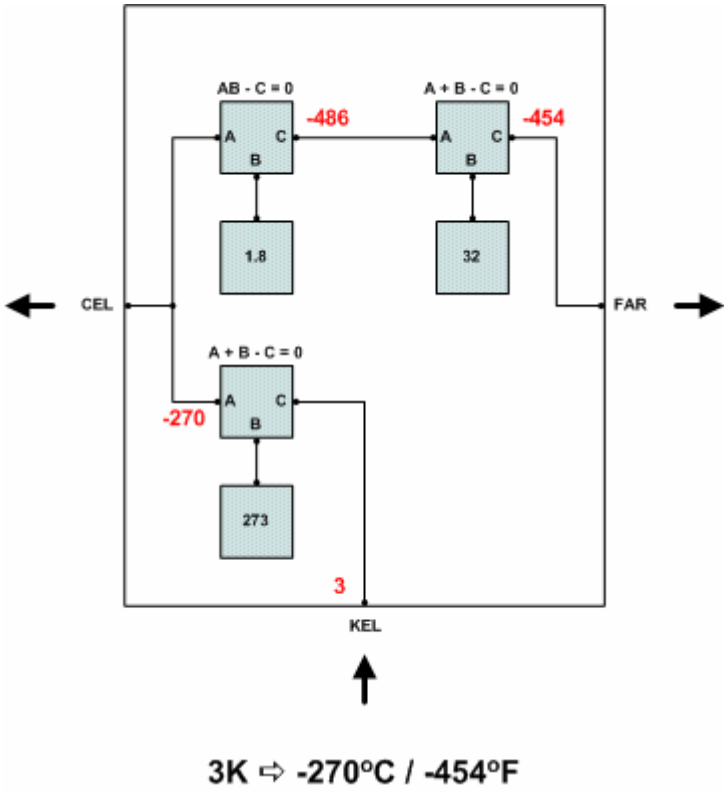




10°C ⇨ 283K / 50°F



32°F ⇨ 0°C / 273K



5.5.Estudo de caso: Mundo dos Besouros

Neste exemplo, simulamos o comportamento de uma comunidade de besouros que seguem fontes de calor. Este problema foi descrito em (Minar, 1996), e o reproduzimos aqui segundo a abordagem de circuitos de objetos.

Considere uma região bidimensional com uma certa propriedade, calor, definida em cada ponto. Um certo número de besouros voa nesta região, procurando locais onde a temperatura é mais amena (cada besouro tem uma temperatura ótima particular). Todavia, as únicas fontes de calor presentes são os próprios besouros; tipicamente, eles emanam uma pequena quantidade de calor, que é dispersa pelo ambiente. Deste modo, a temperatura efetiva de um ponto depende da distribuição de besouros ao redor deste ponto, e do calor emanado por cada um deles.

SWARM (Minar, 1996), uma ferramenta de simulação baseada em sistemas multi-agentes, utiliza o exemplo dos besouros como tutorial. A Ilustração 7 mostra o ambiente da implementação SWARM num dado momento da simulação. Embora os besouros sejam agentes essencialmente reativos, e guiados apenas por seus sensores de calor, podemos observar um comportamento emergente no

sistema: besouros voam em bandos caso precisem de mais calor, ou sozinhos se a temperatura está muito alta.

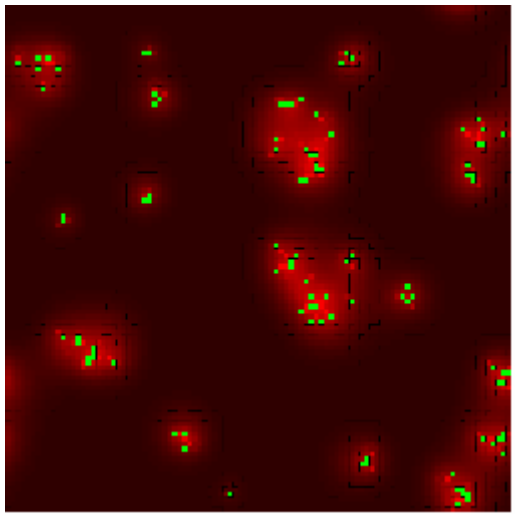


Ilustração 7 Representação gráfica do ambiente.

Passemos agora à implementação do problema dos besouros utilizando circuitos de objetos. Detalhes foram omitidos para melhor expor conceitos-chave. Ilustração 8 mostra o circuito de um besouro. O dispositivo central, BUG BRAIN, controla suas asas e a fonte interna de calor. Ele percebe a quantidade de calor do local através da porta H, e, baseado em algum mecanismo interno de decisão, ordena suas asas a vibrarem de forma a realizar uma certa força (dada por sua projeções vertical e horizontal, dF0 and dF1), e sua fonte de calor a emanar uma certa quantidade de calor (dada por dH).

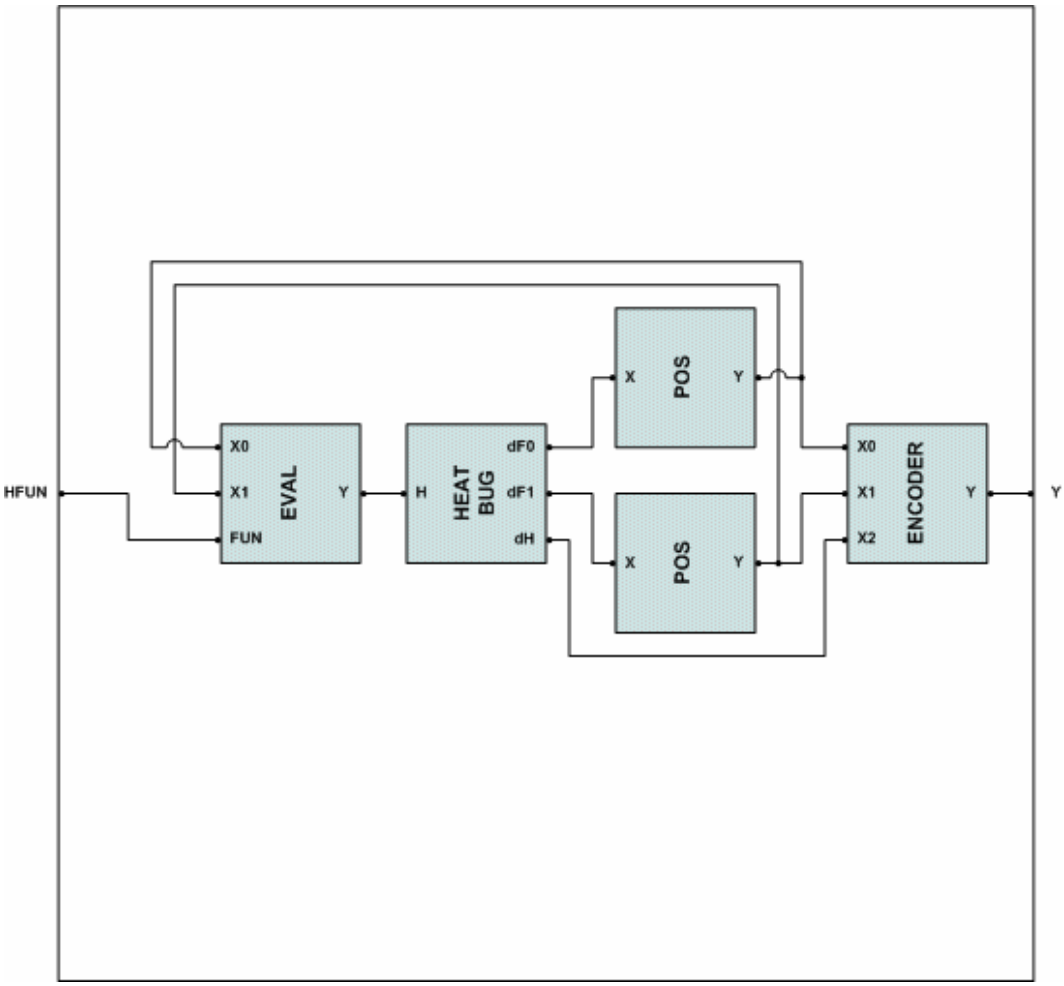


Ilustração 8 Circuito de um besouro.

Os próximos dois dispositivos, denominados POS, acumulam informação sobre a força das asas de forma a atualizar a posição do besouro em relação à direção da força. Então, o dispositivo ENCODER recebe as coordenadas do ponto, juntamente com a quantidade de calor, dH, e os coloca num único vetor. Este vetor pode ser lido a partir do ponto Y do circuito externo. A porta de entrada HFUN é um pouco mais complexa. Ela ilustra o uso de objetos como entidades que também possuem comportamento e não apenas informação. A partir dela, são lidos objetos representando uma função de distribuição de calor. Com esta função, o dispositivo EVAL pode calcular o calor que um vaga-lume está sentindo num dado instante, utilizando como argumentos da função as suas coordenadas, oriundas dos dispositivos POS. A quantidade de calor é emanada na porta de saída, e, em seguida, percebida por BUG BRAIN, que já foi explicado.

De forma a preservar a reusabilidade de EVAL, é necessário que ele trabalhe com uma função genérica, não atrelada a um problema em particular. Nós definimos a interface `Function` que modela tal função genérica. Ela contém um único método, `evaluate`, que aceita um vetor de argumentos da classe `Object`, e retorna o valor que a função admite quando este vetor é utilizado como parâmetro. Em seguida, construímos a classe `HeatFunction`, uma implementação de `Function` cujo trabalho é calcular a quantidade de calor presente num dado ponto.

No nosso exemplo, HFUN percebe um objeto função que é, em última análise, uma instância de `HeatFunction`. Mas, para todos os propósitos, EVAL sabe apenas que está lidando com um objeto função genérica, instância de `Function`, e com uma interface bem-definida.

Estamos agora preparados para explicar o resto do sistema. Ilustração 9 mostra um ambiente com dois besouros. Num dado momento, eles escrevem em Y um vetor contendo sua posição e calor emanado. Os vetores provenientes de todos os besouros são lidos pelo dispositivo integrador, S. Este dispositivo utiliza esta informação para construir um objeto `HeatFunction`, que é então escrito na porta Y. Note que esta porta é conectada às portas HFUN de ambos os besouros, fechando o circuito.

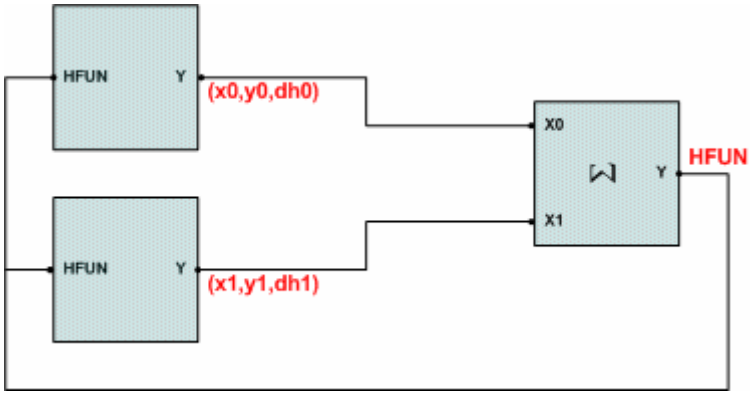


Ilustração 9 Circuito do ambiente. À direita, um dispositivo integrador. À esquerda, dois besouros.

A implementação do ambiente foi então realizada utilizando a arquitetura JOCA. A Ilustração 10 demonstra uma tela de visualização gráfica do ambiente com nove besouros. Todos os besouros são idênticos, possuindo mesma velocidade (constante), nível ótimo de temperatura e capacidade da fonte de calor. Eles foram construídos de tal maneira a capacidade da fonte não é capaz de suprir a necessidade individual de cada besouro. Em outras palavras, um besouro sozinho no ambiente sente "frio". Por isso, é interessante notar como, passados alguns instantes, os besouros já estão mais próximos uns dos outros, de forma a aumentar a temperatura da região.

A inteligência do besouro, isto é, o algoritmo pelo qual ele decide que direção tomar, é baseada no gradiente de temperatura. Dada sua posição corrente, o besouro calcula suas (oito) possíveis posições no instante seguinte, e avalia a temperatura em cada uma delas. O ponto cuja temperatura é mais próxima do nível ótimo é escolhido. É claro que este algoritmo é deficiente e sujeito a ótimos locais, tendo sido escolhido mais por simplicidade do que eficiência.

Ilustração 10 Representação gráfica do ambiente no início da simulação e após algumas iterações.