Sumário

1) Introdução

O objetivo deste documento é apresentar um dispositivo que implementa uma Unidade-Lógico-Aritmética e um sistema de interface para testes.

O dispositivo descrito neste documento é capaz de receber dois vetores de 4 bits e processá-los com 4 operações lógicas (AND, OR, XOR, NOT) e 4 operações aritméticas (incremento, soma, subtração, soma com carry). O resultado é apresentado através de um vetor de saída de 4 bits. Os valores de entrada são gerados de forma dinâmica através de um módulo gerador de entradas. O dispositivo segue um ciclo de 6 segundos: 2 segundos de apresentação do primeiro vetor de entrada; 2 segundos para apresentação do segundo vetor de entrada; 2 segundos para apresentação do resultado calculado pela ULA, correspondente à operação selecionada.

O projeto foi desenvolvido sintetizado e implementado através da interface ISE da empresa Xilinx; simulado no simulador ISim, integrado com a ISE; e testado em hardware através de uma placa de desenvolvimento da Xilinx com um chip FPGA Spartan-3 XC3S700A-FG484.

Para o teste em hardware, os LEDs da placa de desenvolvimento foram configurados para apresentar os resultados do sistema, enquanto as chaves da placa (switches) controlavam a operação realizada pela ULA e os botões de pressão (pushbuttons), em conjunto com uma chave de placa, controlavam as entradas.

O desenvolvimento dos módulos que compõem o dispositivo são detalhadamente apresentados na seção 2 e os resultados das simulações estão descritos na seção 3. O projeto foi desenvolvido em VHDL. Todos arquivos que implementam os módulos descritos estão referenciados no texto e disponíveis no apêndice deste documento. Os nomes utilizados no código fonte para a implementação de cada parte do módulo serão apresentados na descrição de cada módulo em parênteses, diretamente após o seu significado (i.e. [...] a saída do módulo é composta pelo resultado da operação (sinal_resultado) e também[...]) ou diretamente no texto.

2) Descrição da implementação

O projeto será apresentado utilizando uma abordagem bottom up.

2.1) Somador de 4-bits com carry lookahead

O módulo somador executa a operação de soma, necessária para as operações aritméticas. A entidade recebe como entrada dois vetores de 4 bits (A e B) e um bit e carry (carryIn; a ser incluído na soma dos bits menos significativos) e retorna a soma desses valores(S) e dois bits de *carry*, o primeiro correspondente à soma dos bits mais significativos da entrada (carryOut) e o segundo correspondente a soma dos penúltimos (carryOut_1).

Ao invés de usar um módulo somador de um bit e generalizá-lo para n-bits, o somador foi construído de forma que toda lógica booleana é feita diretamente no corpo da arquitetura, sem a necessidade de declarar componentes auxiliares. O *design* foi feito dessa maneira para deixar o código mais simples e organizado, uma vez que, com a implementação do carry lookahead, para cada bit somado deve ser gerado um sinal de *propagate* (P) e um *generate* (G) para o cálculo do carry e retornar, para os n-1 últimos bits (no caso 3 últimos) e para a saída, o resultado do carry (C). Portanto, para todas as operações de soma o código descreve a lógica combinacional diretamente nos vetores, ao invés de separar para cada bits (e.g: A xor B, sendo A e B dois vetores de 4-bits). Devido à diferença da lógica combinacional para cada bit de Carry (na implementação do carry lookahead), essa parte foi descrita bit à bit, assim como a saída, que depende do valor do vetor de carry.

Nomear componentes como A,B,S, P, G e C, em outro contexto, seria uma péssima prática de programação. No caso de um módulo combinacional fixo simples como um somador de 4-bits, a literatura de Sistemas Digitais e Circuitos Lógicos geralmente se refere às entradas e saídas de módulos somadores com essas mesmas etiquetas. Portanto, tomamos como escolha de projeto seguir o modelo da literatura, o que acabou auxiliando na organização do código referente à geração do vetor de carry.

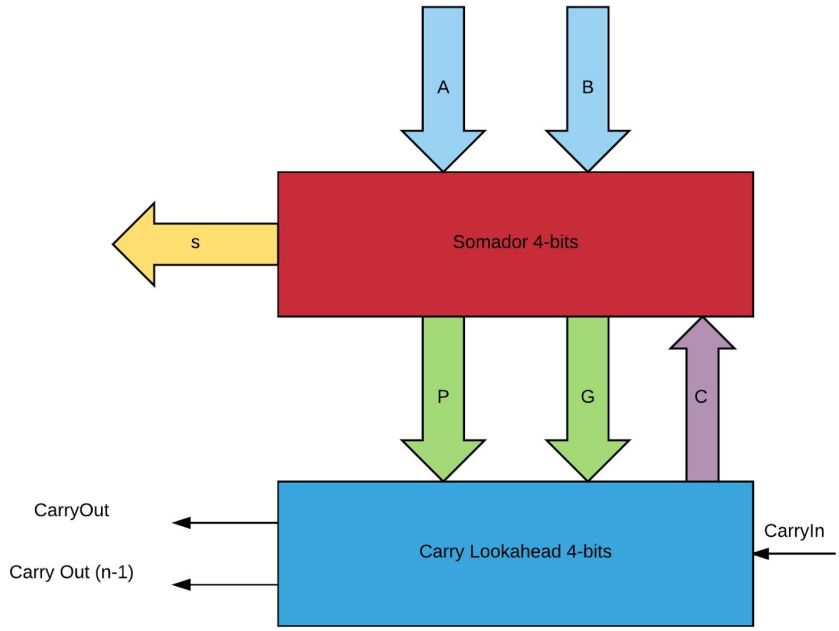


Diagrama de blocos para o somador de 4-bits com *carry lookahead*

Nomear componentes como A,B,S, P, G e C, em outro contexto, seria uma péssima prática de programação. No caso de um módulo combinacional fixo simples como um somador de 4-bits, a literatura de Sistemas Digitais e Circuitos Lógicos geralmente se refere às entradas e saídas de módulos somadores com essas mesmas etiquetas. Portanto, seguiu-se o modelo da literatura, o que acabou auxiliando na organização do código referente à geração do vetor de carry (C).

O código fonte para o somador pode ser encontrado no Apêndice I.

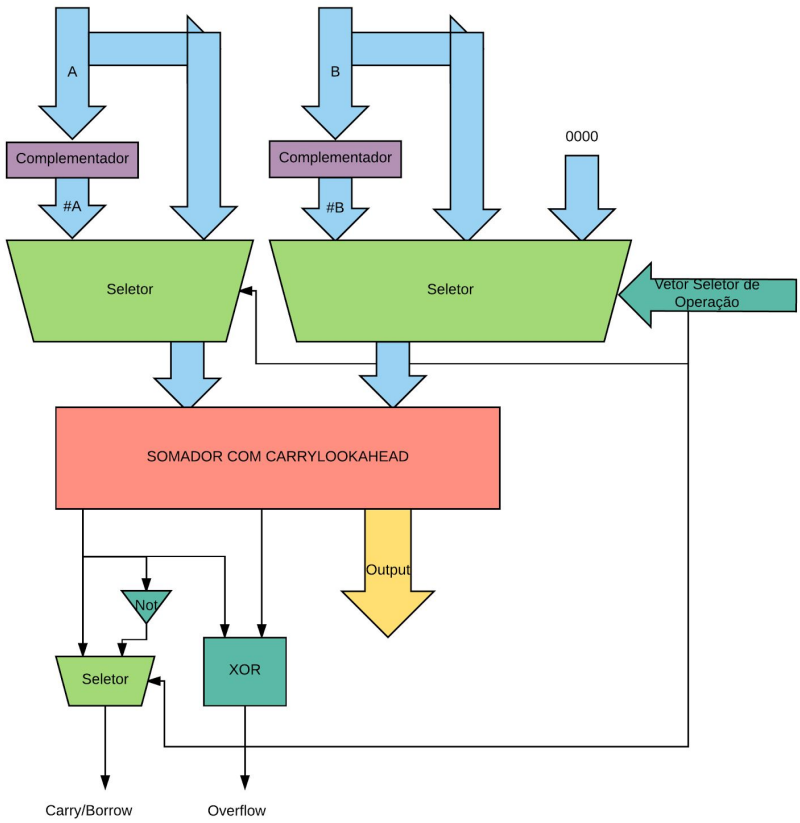
2.2) Unidade Aritmética

Este é o menu das operações aritméticas. Neste módulo, *carryIn*, *entryA* e *entryB* são processados de acordo com a operação selecionada pelo vetor de seleção (selectionVector), e seu resultado é a saída do bloco(output), em conjunto com os flags carry/borrow e overflow. Por exemplo:

No incremento (selectionVector = “011”), *carryIn* é dado como 1, possibilitando a soma unitária, a segunda entrada (*entryB)* é zerada, pois seu valor não interessa, e a primeira entrada (*entryA)* recebe o valor do primeiro vetor de bits dado pelo usuário.

Na subtração (selectionVector = “101”), *carryIn* é dado como 1, a segunda entrada recebe o inverso (por uma operação NOT aplicada ao vetor) do segundo input de vetores dado pelo usuário, possibilitando o complemento a 2 considerando o carryIn, e a primeira entrada *(entryA)* recebe o primeiro input de vetor sem alterações.

Apesar da inversão (selectionVector = “111”) entrar no grupo de operações lógicas, há a possibilidade de realizá-la na Unidade Aritmética ao se aproveitar a operação de inversão (NOT) utilizada para aplicar o complemento de 2. Dessa forma, esta operação foi implementada de forma a inverter a primeira entrada, inserir um vetor de 0s na segunda entrada do somador e zerar o carryIn. Assim obteremos resultado do complemento de 1 da primeira entrada, somado a 0 o que caracteriza a operação.



Vale destacar que os blocos seletores no diagrama acima estão separados meramente para facilitar a visualização. Todas as seleções foram implementadas no código fonte pelo mesmo bloco 'case', implementado em um *process.*

Caso o vetor de seleção (selectionVector) apresente um valor que não corresponde às operações aritméticas (e.g. valores referentes às operações de AND, OR e XOR) o bloco de seleção descrito no código irá selecionar a opção *others*, que corresponde aos valores não designados explicitamente nas outras condições do bloco case. Arbitrariamente, foi escolhido a operação de soma para ser atribuída à saída nessas condições.

No final do módulo, há um mapeamento das entradas e saídas de um componente full adder, que irá receber o valor dos sinais computados no *process* e irá retornar o resultado (output) assim como os bits de carry (carryOut e carryOut_1). Estes últimos são recebidos pelos sinais signal_carryOut e signal_carryOut_1 . Ambos são conectados à uma porta XOR para gerar o bit de overflow (overflow). Para gerar a saída de carry/borrow (carryOut), o sinal recebe valores diferentes dependendo da operação escolhida (ele está presente na lista de sensibilidade do processo) : o sinal recebe carryOut para todas as operações menos a subtração, na qual este recebe carryOut invertido (not carryOut). Devido às requisições do projeto e a escassez de LEDs na placa de desenvolvimento, outros flags como indicador de zero e flag de paridade foram deixados de fora do módulo.

O código fonte para a Unidade aritmética pode ser encontrado no Apêndice II.

2.6) Unidade Lógica

Não há um módulo específico para a Unidade Aritmética. Devido à simplicidade das operações lógicas, as entidades que descrevem as operações de AND, OR e XOR foram declaradas diretamente como componentes no módulo da Unidade Lógica Aritmética.

O código fonte das três operações é extremamente semelhante e simples, recebendo duas entradas (vectorA e vectorB) e retornando o resultado (vectorOut). A arquitetura descreve as operações sendo implementadas nos vetores de 4-bits de entrada.

O código fonte referente à Unidade Lógica pode ser encontrado no Apêndice III.

2.4) Unidade-Lógico-Aritmética

Neste módulo, ocorre a seleção das possíveis operações lógicas designadas pelo vetor de seleção (selectionVector) e as operações da Unidade Aritmética, tratadas com mais detalhes no módulo homônimo. O vetor de seleção possui 3 bits, portanto pode assumir 8 valores distintos, cada um correspondendo à uma operação diferente:

- 000: AND
- 001: OR
- 010: XOR
- 011: INCREMENTO
- 100: SOMA
- 101: SUBTRAÇÃO
- 110: SOMA COM CARRY
- 111: INVERSÃO

Foi arbitrariamente inserido no bloco *others* o resultado de uma soma. Como todos os valores lógicos possíveis foram tratados, o bloco others será selecionado apenas em caso de

uma situação inesperada do sinal selectionVector, como estado de alta impedância no sinal

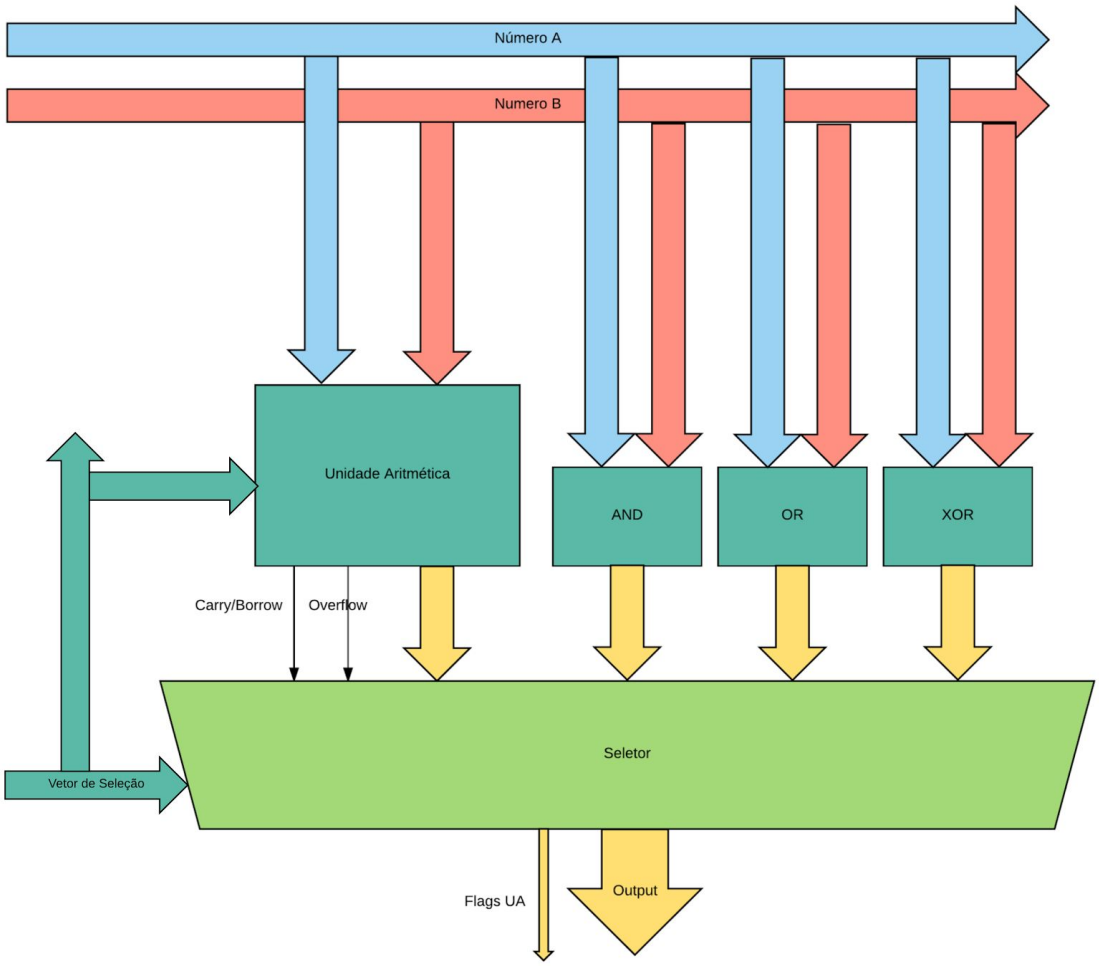


Diagrama de blocos da Unidade-Lógico-Aritmética

Os módulos lógicos e o módulo da Unidade Aritmética estão instanciados no corpo da arquitetura da UL. As entradas referentes aos operandos (inputVectorA e inputVectorB) e sinais de saída (outputAU, outputAND, outputOR, outputXOR, carryOut, overflow) foram mapeados para estas instâncias. O vetor de seleção também é mapeado para a UA de forma que a instância possa selecionar, se for o caso, as operações aritméticas.Todos esses sinais se encontram presentes na lista de sensibilidade de um processo composto de uma declaração *case* que irá selecionar a saída do módulo ULA de acordo com os sinais resultantes dos módulos UA e UL

Operações que não necessitam de um segundo operando (incremento e inversão) são tratadas pela Unidade Aritmética.

O código fonte da Unidade-Lógico-Aritmética pode ser encontrado no Apêndice IV.

2.4) Contador

O módulo em questão é um contador módulo 3 implementado com componentes que simulam o comportamento de um flip flop JK. O módulo flip-flop foi projetado sem as entradas e saídas que não seriam utilizadas no projeto. Dessa forma, Q barrado e preset não aparecem na definição da entity. O comportamento do flip flop é descrito por um *process* que implementa laços condicionais if-else para determinar a saída Q, de acordo com a seguinte tabela de excitação.

J	K	Q
0	0	Q
0	1	0
1	0	1
1	1	#Q

Os sinais, entradas e saídas presentes na arquitetura do módulo contador estão conectados exatamente como o diagrama abaixo.

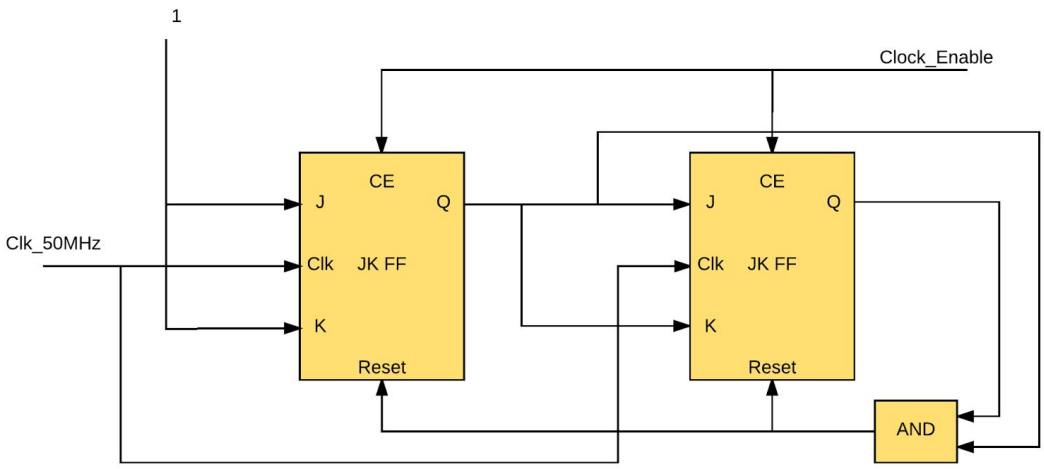


Diagrama de blocos de um contador binário módulo 3 com Flip Flips JK

O código fonte para o flip flop JK e o contador podem ser encontrados no Apêndice V.

2.6)Divisor de frequência

Esse módulo é responsável pela divisão da frequência de clock do sistema (no caso, da placa de desenvolvimento e da simulação) no tempo desejado para a interface do dispositivo. A placa de desenvolvimento utilizada possui um clock de 50 MHz e o objetivo é obter uma frequência de 0.5Hz. Para isso, o módulo foi idealizado com um bloco contador e um bloco comparador.

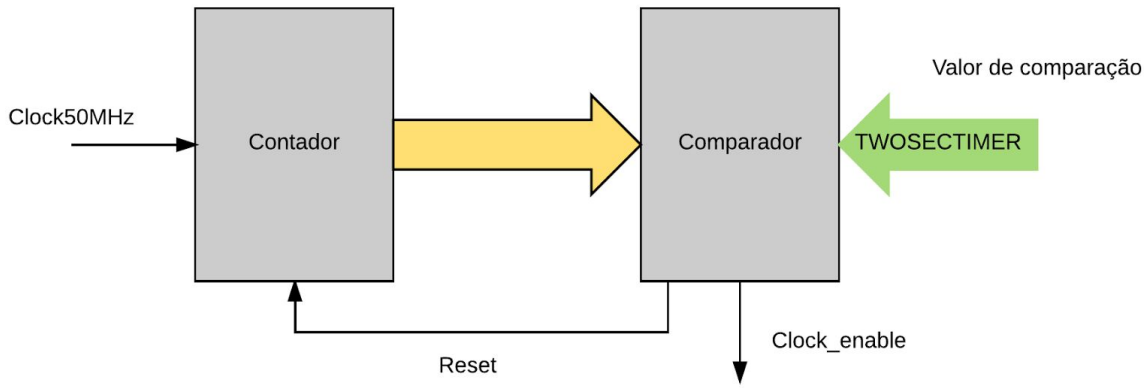


Diagrama de blocos para o divisor de frequência.

A entrada TWOSECTIMER, presente no diagrama acima, corresponde à uma constante no código fonte que determinará o número de ciclos do clock do sistema (clk_50MHz) até o tempo especificado pela constante. A cada clock, o contador incrementa em uma unidade. Quando a contagem atinge o valor especificado, o comparador dispara um pulso de reset para zerar o contador e um pulso clock_enable, que será ligado nos outros módulos sequências do projeto de forma a permitir a passagem de um pulso de clock (clock_50HMz). Após a passagem do pulso, o contador estará zerado e o clock_enable estará desativado até o fim do período de clock. O uso de um enable para o controle do clock do sistema ao invés de utilizar a própria saída do divisor de frequência como clock tem como objetivo evitar dessincronia entre os módulos do sistema.

Ao contrário dos outros módulos sequenciais do projeto, o divisor de frequência não implementa a contagem com o módulo flip flop. Isso se deve ao tamanho da constante necessária para gerar um sinal de enable a cada 2 segundos, o que tornaria o projeto de um contador a partir de flip flops extremamente complexo. Além disso, a implementação utilizada permite maior flexibilidade do módulo, permitindo alternar o período do pulso de enable ao alterar somente o valor da constante TWOSECTIMER. O código, portanto, foi implementado através de um bloco *process* e laços condicionais if-else, que determinavam se seria realizada a contagem ou -- se o valor do contador fosse igual à constante -- se o clock_enable seria ativado e o contador zerado.

O código fonte para o divisor de frequência pode ser encontrado no apêndice VI.

2.7) Gerador de Entradas

O gerador de entradas faz com que seja possível que o usuário entre com os valores desejados para os vetores de entrada da ULA, durante o tempo de execução do programa. O controle de qual número será inserido é dado através da posição do *switch* (em nivel lógico 1 para modificar o valor do primeiro vetor (númeroA) e em nivel lógico 0 para modificar o segundo (número B). O módulo foi projetado tendo em mente os botões de pressão da placa de desenvolvimento, cada um representando um bit do vetor.

Cada bit de entrada (input) está ligado às entradas J e K de dois componentes que simulam o comportamento de um flip-flop JK. O vetor de entrada possui 4 bits, portanto há um total de 8 componentes FF JK instanciados na arquitetura do gerador de entradas: 4 cujas saídas (Q) correspondem aos bits da primeira saída (númeroA) e 4 para a segunda saída (númeroB). Todos as instâncias do componente flip flop estão conectadas ao clock do sistema (clk_50MHz) e a um enable. O sinal de enable irá agir de forma a garantir que apenas 1 dos números seja atualizado por vez -- de acordo com o nível lógico da entrada *switch* -- e apenas mediante a ativação da entrada clock_enable. Isso garante que a atualização dos valores seja feita de forma síncrona com o resto do dispositivo, além de evitar incertezas nos resultados do módulo devido à trepidações de algum componente mecânico ligado à entrada (input). Para obter esse resultado, o pulso do clock_enable deve ficar ativo por um intervalo

de tempo muito curto e desativado por um longo período de tempo comparado ao clock do sistema. Como será descrito na seção sobre o módulo menu, o gerador de entrada foi conectado ao clock_enable do divisor de frequência, garantindo que o flip-flop irá processar a entrada a cada 2 segundos e apenas por um intervalo de tempo correspondente à um período do clock do sistema.

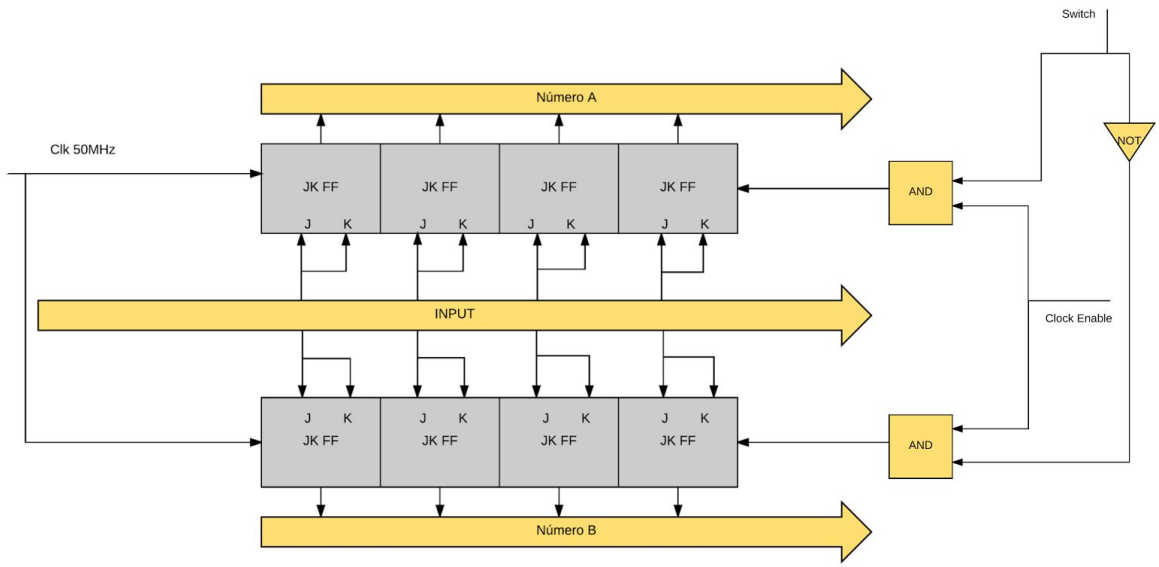


Diagrama de blocos do Gerador de Entradas

Por esse motivo, os bits de *input* referentes aos bits que desejam ser alterados devem permanecer em nível alto durante o pulso do clock_enable para que os valores do número desejado sejam alterados.

Cada bit da entrada ligada simultaneamente às entradas J e K dos flip-flops permite a inversão (toggle) de estados da saída quando a entrada está em nível alto e a manutenção dos estados de saída quando está em nível baixo.

O motivo da utilização do módulo JK sobre um módulo de um flip-flop tipo T (toggle) se deve à possibilidade de reutilizar um dos módulos utilizados para a contrução do contador, além da facilidade de atribuir uma função toggle sobre módulo flip-flop JK, como pode ser observado no diagrama de blocos deste módulo e no código presente no Apêndice VII.

2.8) Menu

O módulo do menu realiza a integração dos módulos sequenciais (gerador de entrada, divisor de frequência, contador) e a Unidade-Lógica-Aritmética, além de receber as entradas e retornar saídas para/de um ambiente externo (no caso, os botões da placa de desenvolvimento e os valores definidos nos programas de *test bench* utilizados para as simulações).

Este módulo é responsável por receber os bits referentes ao input e *switch* a serem passados para o gerador de entradas, que por sua vez irá gerar os operandos para a ULA, que então irá retornar o valor do resultado da operação (alu_output) -- juntamente com os flags de carry e overflow (au_flags(1) -> Carry/Borrow; au_flags(0) -> overflow) -- para ser apresentado no momento adequado. O vetor de seleção (selectionVector) entregue ao módulo ULA também é entrada do módulo menu.

A entrada clk_50MHz recebe o clock do sistema no qual o dispositivo está operando. No caso da placa de desenvolvimento utilizada e da simulação no ISim, a frequência utilizada foi de 50MHz, como está indicado na própria etiqueta da entrada. O clock será atribuído para as instâncias dos componentes do divisor de frequência, gerador de entradas e contador, sendo que esses dois últimos ainda receberão um sinal clock_enable (enable) do divisor de frequência que, nesta implementação, é ativado por 1 período de clock a cada 2 segundos. Portanto, a cada 2 segundos, o contador irá contar 1 unidade, de 0 até 2. Esta contagem (counter) está incluída na lista de sensibilidade de um processo que contém um bloco case. Este bloco irá selecionar a partir dos valores do contador (00, 01 e 10) um bloco de operações à ser feito:

- caso 00: O bloco irá ligar a saída do módulo menu (signal_menuOutput) ao primeiro operando (numeroA) gerado pelo gerador de entradas. Os mostradores de carry e overflow são zerados pois este caso não apresenta o resultado da operação. O sinal presentation_flags corresponde à um indicador do que está sendo apresentado para fora do sistema naquele momento. No caso ‘00’, o primeiro operando está sendo mostrado, e o indicador correspondente é ‘10’.

- caso 01: O bloco irá ligar o sinal de saída do módulo menu (signal_menuOutput) ao segundo operando (númeroB). Os mostradores de carry e overlfow permanecem zerados. O sinal presentation_flag recebe o indicador ‘01’.
- caso 10: O sinal de saída (menuOutput) recebe o resultado da operação da ULA, assim como o sinal signal_au_flags recebe os sinais de carry e overflow. O sinal presentation_flag recebe o indicador ‘11’.
- caso others: sinal inesperado no contador, como o valor 11 ou um sinal em alta impedãncia. Foi decidido arbitrariamente que o comportamento nesse caso é apagar (00) todas as flags (au e *presentation flags)* e manter a saída numérica do estado anterior ao sinal recebido.

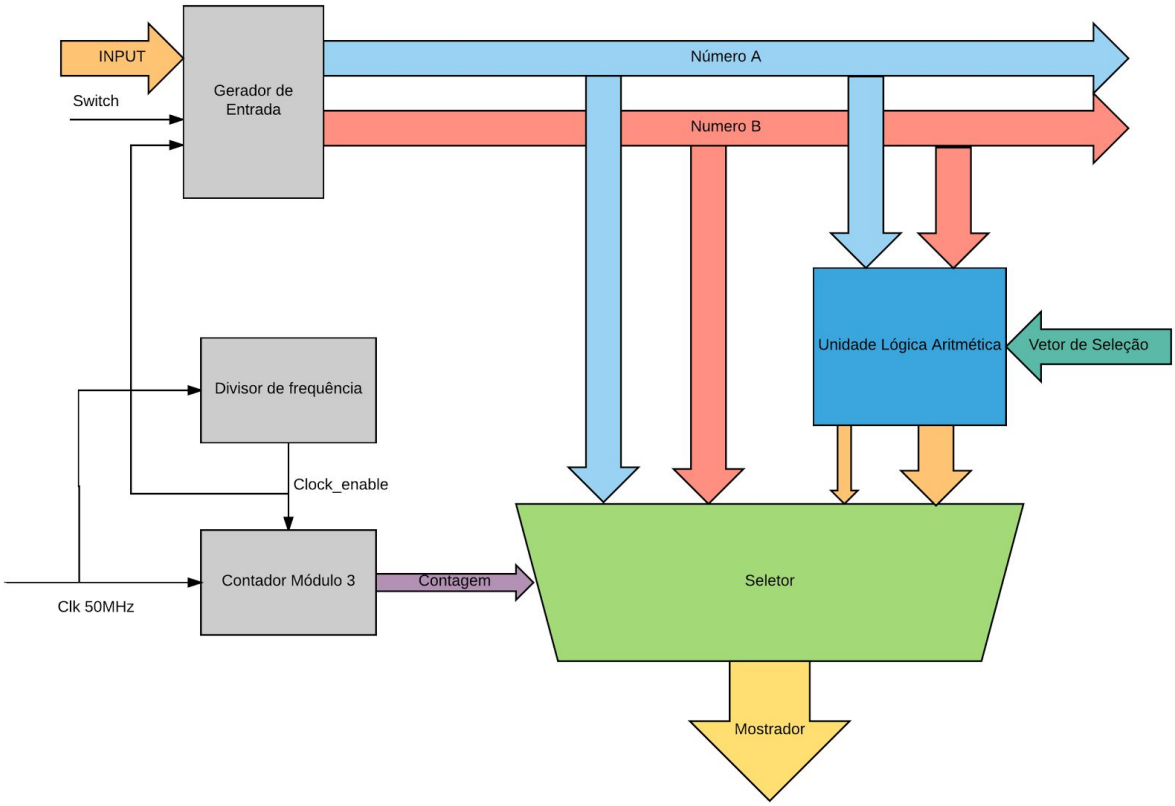


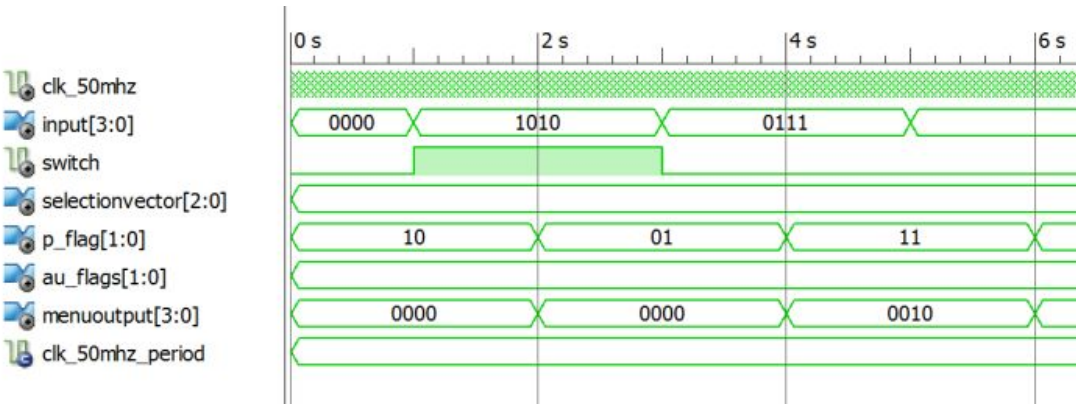
Diagrama de blocos do menu

Vale ressaltar que a saída apresentada no diagrama de blocos acima inclui tanto o mostrador dos operandos e resultados quanto os flags indicadores e flags de carry e borrow.

O código para o módulo menu pode ser encontrado no Apêndice VIII.

3)Resultados

Devido ao modo de operação do gerador de entradas, todas as simulações realizadas utilizam os primeiros 6 segundos para gerar os operandos. Alterando o valor no switch ligado ao gerador de entradas, e inserindo um valor na entrada *input* obtemos o diagrama abaixo.

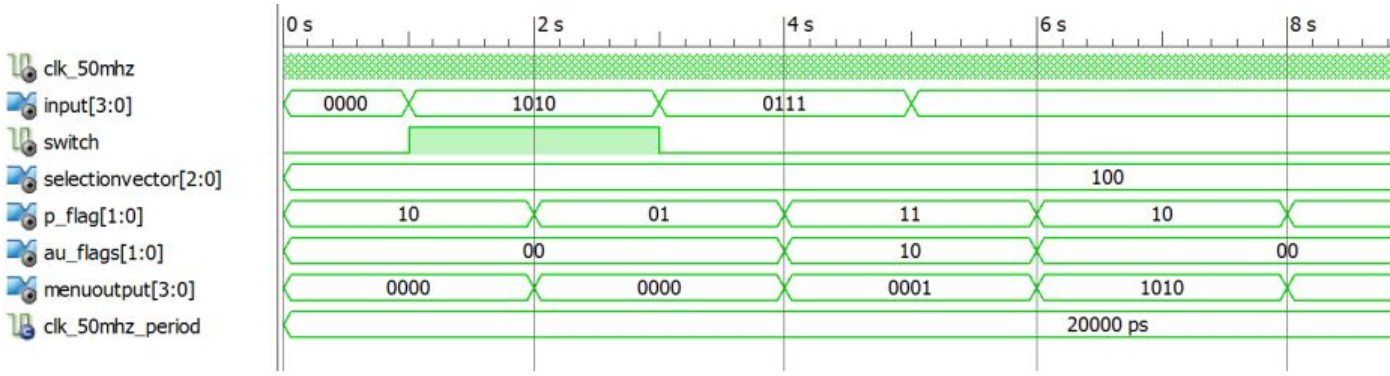


Observe que foram necessários dois pulsos do clock_enable para inicializar os operandos. Sendo assim, apesar do resultado da operação aparecer entre 4 e 6 segundos, os valores inseridos para os operandos irão ser apresentados apenas após 3 pulsos de clock_enable (6 segundos).

Portanto, os resultados obtidos nas simulações para cada operação serão mostrados abaixo a partir do início do segundo ciclo de operação, ou seja, a partir do segundo 6. Para todas as operações, os valores “1010” e “0111” foram utilizados como operandos.

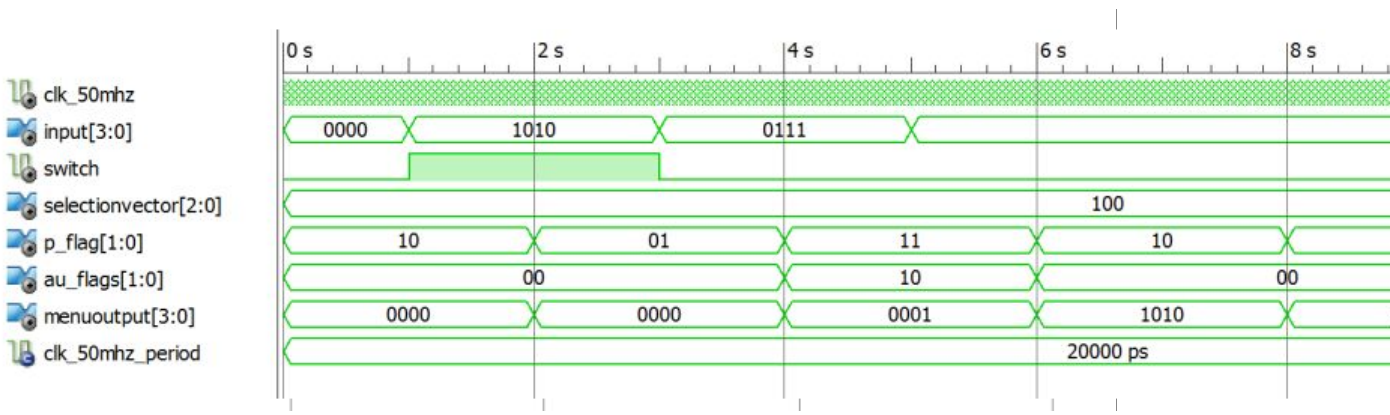
3.1)And

O diagrama a seguir apresenta a simulação realizada para a operação AND.

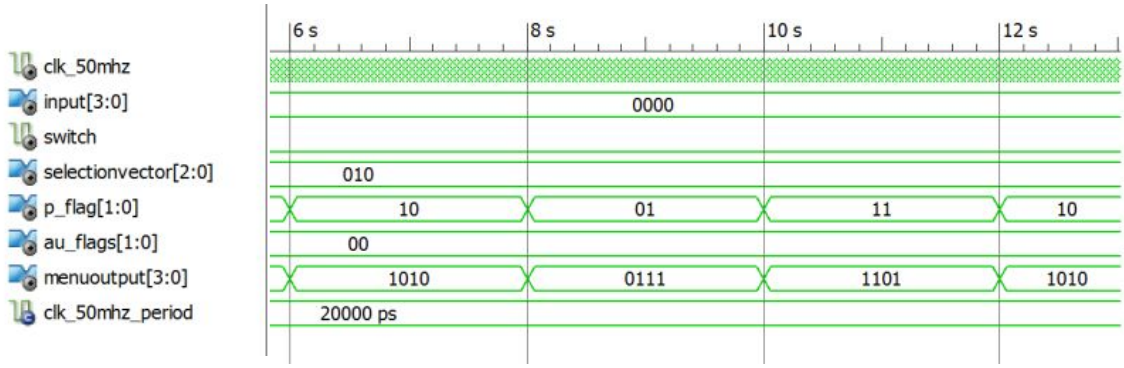


A entrada selectionVector se encontra com o valor “000” para configurar a operação. Os operandos aparecem como saída em menuOutput por 2 segundos cada, seguidos do resultado da operação entre eles. A saída p_flag muda de valor à cada 2 segundos, indicando o que está sendo apresentado em menuOutput: operando A; operando B; resultado. Como as operações lógicas não fazem uso dos flags da unidade aritmética, au_flags permanece em “00” durante toda a operação. As outras operações lógicas repetem o mesmo processo, apenas com um valor diferente em selectionVector(indicando a operação) e um resultado diferente a partir do segundo pulso de clock_enable.

3.2)Or

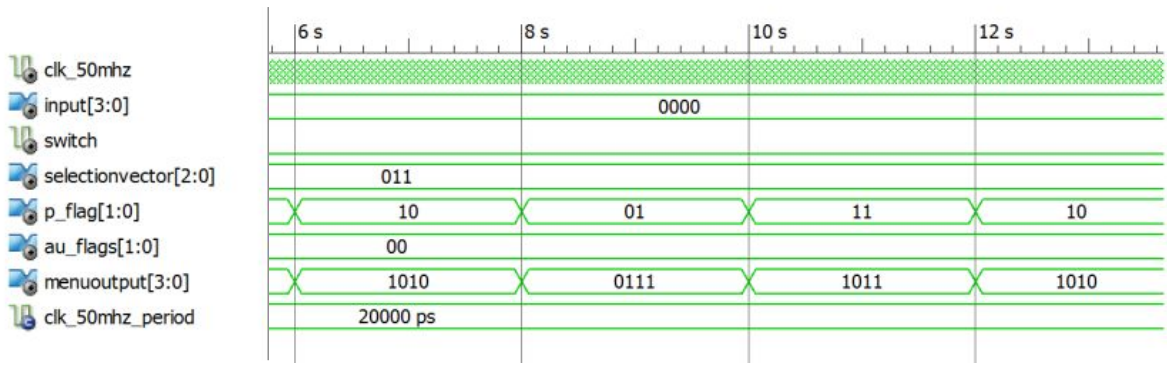


3.3)Xor



3.4)Incremento

O seguinte diagrama representa um incremento sobre o operando A.

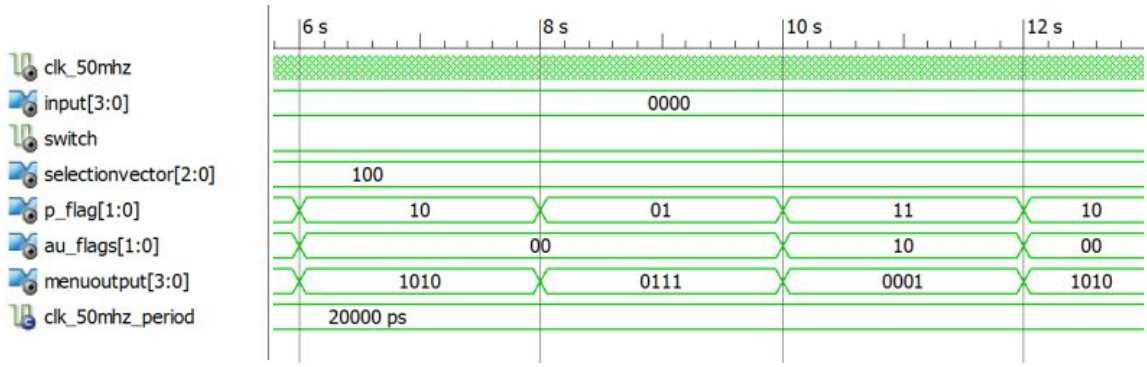


Como a operação de incremento necessita de apenas um único operando, o segundo número é desconsiderado.

As flags da unidade aritmética operam em incremento. Não obstante, o resultado apresentado não configura um caso de soma binária em que há carry e/ou overflow.

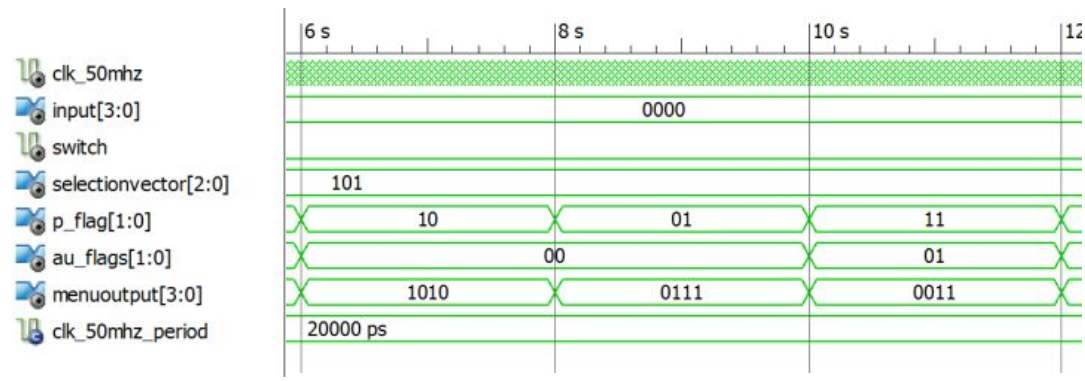
3.5)Soma

O seguinte diagrama representa uma soma com carryInput em ‘0’. As flags da unidade aritmética operam em soma, com o flag carry/borrow atuando como carry. O resultado da soma binária entre os dois operandos configura um caso onde há carry, o que pode ser observado pela saída au_flags.



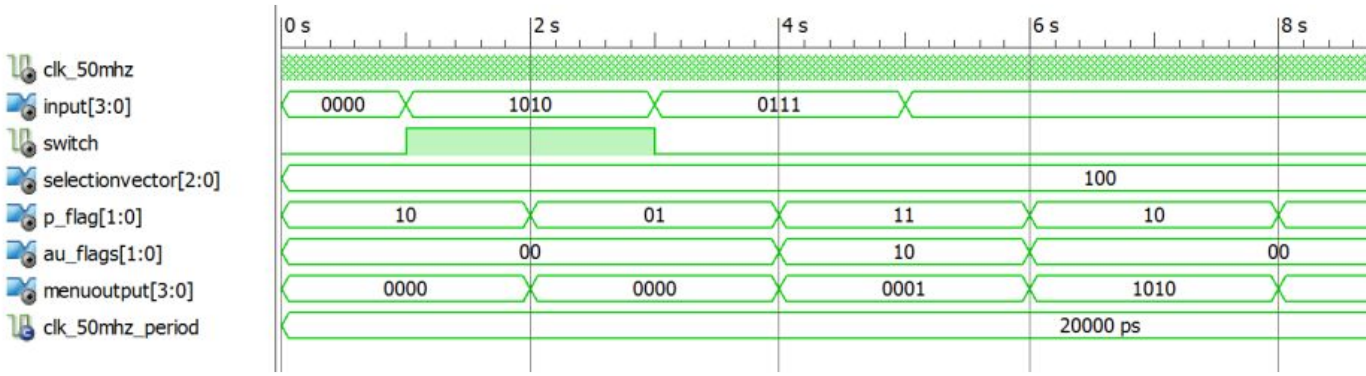
3.6)Subtração

O seguinte diagrama representa uma subtração entre os operandos. As flags da unidade aritmética operam em subtração, com o flag carry/borrow atuando como borrow. O resultado da soma binária entre os dois operandos configura um caso onde há overflow, o que pode ser observado pela saída au_flags.



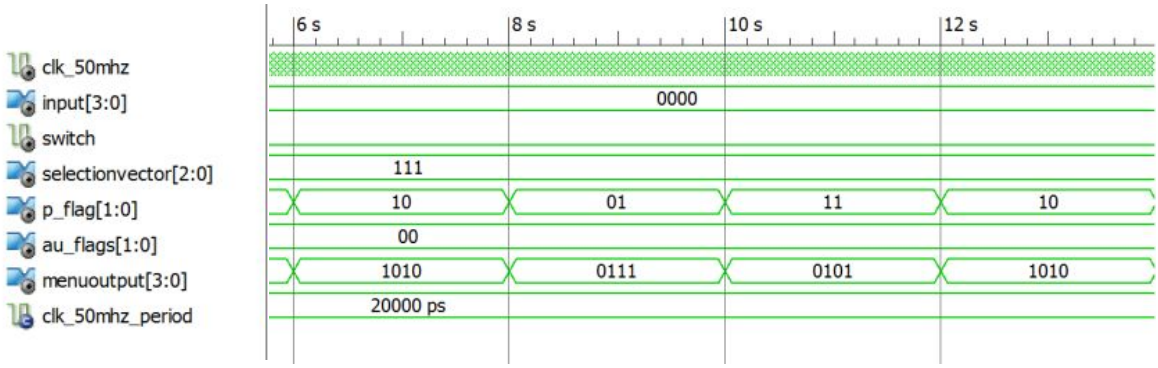
3.7)Soma com carry

O diagrama a seguir representa uma soma com carryInput em ‘1’. Vale ressaltar que esse resultado condiz com o resultado da soma sem carry acrescido de uma unidade. As flags da unidade aritmética operam em soma, com o flag carry/borrow atuando como carry. O resultado da soma binária entre os dois operandos configura um caso onde há carry, o que está de acordo com a operação, visto que o resultado da soma sem carry também apresentou o flag ativo.



3.8)Inversão

O próximo diagrama corresponde à operação de NOT (inversão) e se comporta de forma semelhante aos diagramas das outras operações lógicas, com exceção da quantidade de operandos utilizados para o cálculo do resultado. Como a operação de not necessita de apenas um único operando, o segundo número é desconsiderado.



4) Conclusões

O dispositivo foi testado em hardware e em simulação computacional, apresentando todos os resultados de acordo com as operações configuradas. A utilização de uma linguagem de descrição de máquina facilitou em grande parte a confecção dos módulos, permitindo que a Unidade Aritmética e o sistema de menu/testes fossem projetados, testados e implementados de forma eficiente. Além disso, permitiu que diferentes métodos de construção dos módulos fossem testados de forma a selecionar o que melhor se enquadrasse nos requisitos do projeto e nas limitações de hardware.

APÊNDICES