MD5
Statement Of The Work

Autores: PEM

Última Revisão: 2 de Fevereiro de 2017

Sumário

Lista de Tabelas

Lista de Figuras

Controle de Versão

Revision	Date	Author	Revised	Comments
1.0	25/01/2017	Agripino	-	Initial Version

1 Introdução

Em segurança da informação, autenticação de mensagens é uma técnica essencial para verificar se as mensagens recebidas vieram da fonte alegada e que não foi alterada. Um elemento principal dos esquemas de autenticação é o uso de códigos de autenticação de mensagem (*Message Authenticaton Code* - MAC). Uma técnica para produzir um MAC é baseada no uso de função *hash*.

Uma função *hash* é qualquer função que pode ser usada para mapear dados de valor arbitrário em dados de tamanho fixo. Funções desse tipo são relacionadas a (e geralmente confundidas com) *checksums*, códigos de correção de erro, compressão com perda e *ciphers*, por exemplo. Embora esses conceitos sobreponham-se, cada qual tem seus usos e solicitações e são projetados e otimizados diferentemente.

Uma função *hash* criptográfica é um algoritmo matemático que mapeia dados de tamanho arbitrário para um *string* de *bits* de tamanho fixo, o qual é projetado para também ser uma função unidirecional. Funções *hash* são muito importantes e comuns em primitivas de criptografia. Sua aplicação primária é seu uso em conjunto com criptosistemas de chave pública em esquemas de assinatura digital. Também é um bloco de construção básica de chaves secretas em autenticação de mensagem.

O dado de entrada é chamado de "mensagem" e a saída (o "valor de *hash"* ou simplesmente *hash*) é frequentemente chamado de "mensagem sumária" (*message digest*) ou simplesmente "sumário" (*digest*).

Uma função *hash* criptográfica ideal tem cinco principais propriedades:

- É determinístico, assim a mesma mensagem sempe resulta no mesmo *hash*;
- É rápido de calcular o valor de *hash* para qualquer mensagem dada;
- É impraticável gerar a mensagem a partir do seu valor de *hash*, a não ser tentando todas as mensagens possíveis;
- Uma mudança mínima na mensagem deve mudar o valor de *hash* tão consideravelmente que o novo *hash* parece não-correlacionado com o anterior;
- É impraticável achar duas mensagens diferentes com o mesmo valor de *hash*.

A seguir será descrito o padrão MD5, em seguida o protocolo para implementação da interface de barramento AMBA APB e, por fim, a especificação do projeto a ser desenvolvido.

2 MD5

MD5 é uma função *hash* criptográfica desenvolvida pelo professor Ronald Rivest do MIT. Gera uma valor de *hash* de 128 *bits* conhecido por "mensagem sumária" (*message digest*). Esse padrão é planejado para aplicações de assinatura digital, onde um arquivo deve ser comprimido de maneira segura antes de ser encriptografado com uma chave privada sob um criptosistema de chave pública tal como RSA.

O MD5 pode ser descrito em dois estágios: pré-processamento e cálculo do *hash*.

2.1 Pré-processamento

Envolve preencher a mensagem (*padding the message*), decompor a mensagem preenchida (*parsing the padded message*) em blocos de m *bits* e configurar os valores de inicialização a serem usados no cálculo do *hash*.

2.1.1 Preenchimento da Mensagem

O propósito desse preenchimento é garantir que a mensagem preenchida seja um múltiplo de 512 *bits*. Suponha que o comprimento da mensagem M seja l *bits*. Anexe o *bit* 1 ao final da mensagem, seguido de k zero *bits*, onde k é o menor valor, não negativo, solução da equação $l + 1 + k = 448$. Depois, anexe um bloco de 64 *bits* que é igual ao número l expresso usando uma representação binária.

Exemplo: a mensagem (8 *bits* ASCII) "abc" tem comprimento $8 \times 3 = 24$. Assim, a mensagem é preenchida com o *bit* 1, depois $448 - (24 + 1) = 423$ zero *bits* e o comprimento da mensagem, para tornar-se por fim a mensagem preenchida de 512 *bits*.

$$\underbrace{01100001}_{a}\ \underbrace{01100010}_{b}\ \underbrace{01100011}_{c}\ 1\ \overbrace{000\ldots000}^{423}\ \overbrace{00\ldots\underbrace{011000}_{l=24}}^{64}$$

2.1.2 Decompondo a Mensagem

A mensagem e seu preenchimento são decompostos em blocos de $N\,512$ *bits*. Equivalentemente, essa mensagem possui um comprimento que é um múltiplo exato de dezesseis palavras de 32 *bits*. Seja $M[0, \ldots, N-1]$ a denotação das palavras da mensagem resultante, onde N é um múltiplo de 16.

2.1.3 Configurando o Valor Inicial do *Hash*

Antes do estágio do cálculo do *hash* iniciar, o valor inicial do *hash* deve ser configurado. Para o MD5, um *buffer* de quatro palavras (A, B, C e D) é usado para calcular a mensagem sumária. A , B , C e D são registradores de 32 *bits*. Esses registradores são inicializados com os seguintes valores (representados em hexadecimal e com os *bytes* menos significativos primeiro):

$$A = 0x01234567$$

$$B = 0x89abcdef$$

$$C = 0xfedcba98$$

$$D = 0x76543210$$

2.2 Cálculo do *Hash*

O cálculo do *hash* produz um arranjo de mensagens a partir da mensagem preenchida e usa esse arranjo, em conjunto com funções, constantes e operações para iterativamente produzir uma série de valores de *hash*. O valor de *hash* final produzido pelo cálculo é usado para determinar a mensagem sumária. Na Tabela 1 pode ser visto as propriedades básicas do algoritmo do MD5.

Tabela 1: Propriedades do MD5

Tamanho da mensagem (*bits*)	Tamanho do bloco (*bits*)	Tamanho da palavra (*bits*)	Tamanho da mensagem sumária (*bits*)
$< 2^{64}$	512	32	128

MD5 utiliza uma sequência de funções lógicas. Cada função opera em três palavras de 32 *bits* cada (x, y e z) e produz uma palavra de 32 *bits* como saída. As funções são definidas como:

$$f_t(x,y,z) = \begin{cases} F(x,y,z) = (x \cdot y) + (\overline{x} \cdot z) & 1^a\, rodada \\ G(x,y,z) = (x \cdot z) + (y \cdot \overline{z}) & 2^a\, rodada \\ H(x,y,z) = x \oplus y \oplus z & 3^a\, rodada \\ I(x,y,z) = y \oplus (x + \overline{z}) & 4^a\, rodada \end{cases}$$

O MD5 utiliza uma tabela ($T[63:0]$) de sessenta e quatro palavras constantes de 32 *bits* cada, obtidas da função seno. Seja $T[i]$ o $i-ésimo$ elemento da tabela, que é igual a parte interia da equação $2^{32} \times abs(sin(i+1))$, onde i está em radiano. Os elementos da tabela são dados a seguir:

$$T[3:0] = \{0xc1bdceee,\ 0x242070db,\ 0xe8c7b756,\ 0xd76aa478\}$$

$$T[7:4] = \{0xfd469501,\ 0xa8304613,\ 0x4787c62a,\ 0xf57c0faf\}$$

$$T[11:8] = \{0x895cd7be,\ 0xffff5bb1,\ 0x8b44f7af,\ 0x698098d8\}$$

$$T[15:12] = \{0x49b40821,\ 0xa679438e,\ 0xfd987193,\ 0x6b901122\}$$

$$T[19:16] = \{0xe9b6c7aa,\ 0x265e5a51,\ 0xc040b340,\ 0xf61e2562\}$$

$$T[23:20] = \{0xe7d3fbc8,\ 0xd8a1e681,\ 0x02441453,\ 0xd62f105d\}$$

$$T[27:24] = \{0x455a14ed,\ 0xf4d50d87,\ 0xc33707d6,\ 0x21e1cde6\}$$

$$T[31:28] = \{0x8d2a4c8a,\ 0x676f02d9,\ 0xfcefa3f8,\ 0xa9e3e905\}$$

$$T[35:32] = \{0xfde5380c,\ 0x6d9d6122,\ 0x8771f681,\ 0xfffa3942\}$$

$$T[39:36] = \{0xbebfbc70,\ 0xf6bb4b60,\ 0x4bdecfa9,\ 0xa4beea44\}$$

$$T[43:40] = \{0x04881d05,\ 0xd4ef3085,\ 0xeaa127fa,\ 0x289b7ec6\}$$

$$T[47:44] = \{0xc4ac5665,\ 0x1fa27cf8,\ 0xe6db99e5,\ 0xd9d4d039\}$$

$$T[51:48] = \{0xfc93a039,\ 0xab9423a7,\ 0x432aff97,\ 0xf4292244\}$$

$$T[55:52] = \{0x85845dd1,\ 0xffeff47d,\ 0x8f0ccc92,\ 0x655b59c3\}$$

$$T[59:56] = \{0x4e0811a1,\ 0xa3014314,\ 0xfe2ce6e0,\ 0x6fa87e4f\}$$

$$T[63:60] = \{0xeb86d391,\ 0x2ad7d2bb,\ 0xbd3af235,\ 0xf7537e82\}$$

O algoritmo do MD5 pode ser usado para obter o *hash* de uma mensagem, M, tendo um comprimento de l *bits*, onde $0 \leq l < 2^{64}$. O resultado final do MD5 é uma mensagem sumária de 128 *bits*.

2.2.1 Algoritmo

Todas as adições são realizadas em módulo de 2^{32}. O processamento consiste de quatro estágios similares, nomeados de rodadas. Cada rodada é composta de dezesseis operações similares baseadas em uma função não linear f_t, adição modular e rotação para esquerda. A seguir o algoritmo é mostrado:

```
// Process each 16-word block.
For i = 0 to N/16-1 do

        // Copy block i into X.
        For j = 0 to 15 do
                Set X[j] to M[i*16+j].
        end

        // Save A as AA, B as BB, C as CC, and D as DD.
        AA = A
        BB = B
        CC = C
        DD = D

        // Round 1.
        // Let [abcd k s i] denote the operation
        // a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s).
        // Do the following 16 operations.
        [ABCD  0 7  1] [DABC  1 12  2] [CDAB  2 17  3] [BCDA  3 22  4]
        [ABCD  4 7  5] [DABC  5 12  6] [CDAB  6 17  7] [BCDA  7 22  8]
        [ABCD  8 7  9] [DABC  9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
        [ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

        // Round 2.
        // Let [abcd k s i] denote the operation
        // a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s).
        // Do the following 16 operations.
        [ABCD  1 5 17] [DABC  6 9 18] [CDAB 11 14 19] [BCDA  0 20 20]
        [ABCD  5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA  4 20 24]
        [ABCD  9 5 25] [DABC 14 9 26] [CDAB  3 14 27] [BCDA  8 20 28]
        [ABCD 13 5 29] [DABC  2 9 30] [CDAB  7 14 31] [BCDA 12 20 32]

        // Round 3.
        // Let [abcd k s t] denote the operation
        // a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s).
        // Do the following 16 operations.
```

```
[ABCD  5 4 33] [DABC  8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD  1 4 37] [DABC  4 11 38] [CDAB  7 16 39] [BCDA 10 23 40]
[ABCD 13 4 41] [DABC  0 11 42] [CDAB  3 16 43] [BCDA  6 23 44]
[ABCD  9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA  2 23 48]

// Round 4.
// Let [abcd k s t] denote the operation
//   a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s).
// Do the following 16 operations.
[ABCD  0 6 49] [DABC  7 10 50] [CDAB 14 15 51] [BCDA  5 21 52]
[ABCD 12 6 53] [DABC  3 10 54] [CDAB 10 15 55] [BCDA  1 21 56]
[ABCD  8 6 57] [DABC 15 10 58] [CDAB  6 15 59] [BCDA 13 21 60]
[ABCD  4 6 61] [DABC 11 10 62] [CDAB  2 15 63] [BCDA  9 21 64]

/* Then perform the following additions. (That is increment each
   of the four registers by the value it had before this block
   was started.) */
A = A + AA
B = B + BB
C = C + CC
D = D + DD
```

end

Após repetir os passos anteriores num total de N vezes, a mensagem sumária de 128 *bits* resultante é A, B, C, D. Isso é, começa-se com o byte menos significativo de A e termina com o byte mais significativo de D.

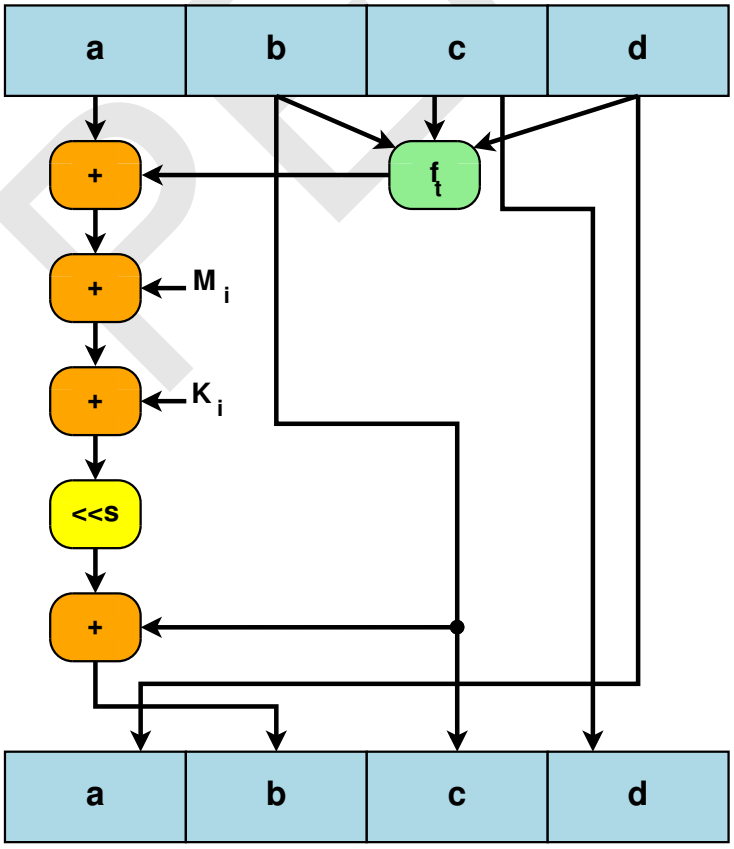

Figura 1: Ilustração das operações realizadas nas rodadas

3 Protocolo de Interface AMBA APB

O protocolo AMBA (*Advanced Microcontroller Bus Architecture*) é um padrão aberto, especificação de interconexão *on-chip* para conexão e gerenciamento de blocos funcionais em um SoC (*System-on-Chip*). Permite o desenvolvimento de projetos multi-processador com um grande número de controladores e periféricos. O AMBA promove o reuso de projeto definindo-se um barramento comum para módulos SoC usando especificações para ACE, AXI, AHB, APB e ATB (tipos do protocolo AMBA).

Um importante aspecto de um SoC não é somente quais componentes ou blocos ele comporta, mas também como eles estão interconectados. O protocolo AMBA é uma solução para a interface. Os objetivos do AMBA são:

- Facilitar o desenvolvimento *right-first-time* de produtos microcontroladores embarcados com um ou mais CPUs, GPUs ou processadores de sinais;
- Ser independente de tecnologia, permitindo o reuso do IP (*Intelectual Property*), periféricos e macrocélulas de sistema sobre diversos processos de circuitos integrados;
- Encorajar projeto de sistemas modulares para melhorar a independência do processador e o desenvolvimenro da reusabilidade do periférico e bibliotecas de IP do sistema;
- Suportar alto desempenho e baixo comsumo na comunicação dentro do circuito integrado.

O APB (*Advanced Peripheral Bus*) é parte da família de protocolo AMBA. Define uma interface de baixo custo que é otimizada para baixo consumo de energia e reduzida complexidade de interface. Não é do tipo *pipelined* e é usado para conectar periféricos de baixa largura de banda que não requer um alto desempenho.

O APB relaciona uma transição de sinal à borda de subida do sinal de relógio (*clock*), para simnplificar a integração dos periféricos APB em qualquer fluxo de projeto. Cada transferência toma no mínimo dois ciclos de relógio.

3.1 Sinais do AMBA APB

A Tabela 2 descreve os sinais presentes no protocolo AMBA APB. Observe que o protocolo possui dois barramentos de dados independentes, um para leitura dos dados e outro para escrita dos dados. Devido esses barramentos de dados não possuirem seus próprios sinais de handshake, não é possível as transferências de dados ocorrerem em ambos os barramentos ao mesmo tempo.

Sinal	Fonte	Descrição
PCLK	Fonte de relógio	Sinal de relógio. A borda de subida de PCLK sincroniza todas as transferências no APB
PRESETn	Barramento do sistema	Sinal de reset. Esse sinal é do tipo nível baixo ativo. É normalmente conectado diretamente ao sinal de reset do barramento de sistema
PADDR	Canal APB	É o barramento de endereço do APB. Pode ser de até 32 *bits* de largura e é conduzido pela unidade de canal do barramento periférico
PSELx	Canal APB	A unidade de canal APB gera esse sinal para cada periférico escravo do barramento. Indica que o dispositivo escravo é selecionado e que uma transferência de dados foi requerida. Há um sinal de PSELx para cada escravo
PENABLE	Canal APB	Esse sinal indica o segundo e subsequentes ciclos de uma transferência APB
PWRITE	Canal APB	Esse sinal indica um acesso de escrita quando ”1” ou um acesso de leitura quando ”0”
PWDATA	Canal APB	Esse barramento é alimentado pela unidade de canal do barramento periférico durante os ciclos de escrita quando PWRITE é ”1”. Esse barramento pode ter até 32 *bits* de largura
PREADY	Interface de escravo	O escravo usa esse sinal para extender uma transferência APB
PRDATA	Interface de escravo	O escravo selecionado alimenta esse barramento durante os ciclos de leitura quando PWRITE é ”0”. Esse barramento pode ter até 32 *bits* de largura.

Tabela 2: Lista de sinais do APB

3.2 Transferências de escrita

Observação: Recomenda-se que os sinais de endereço e escrita não mudem logo após uma transferência, mas que permaneçam estáveis até outro acesso ocorrer. Isso reduz o consumo de energia.

Esta seção descreve os seguintes tipos de transferência de escrita:

- Sem estados de espera
- Com estados de espera

3.2.1 Sem Estados de Espera

Pode-se observar na Figura 2 uma transferência de escrita básica sem estados de espera.

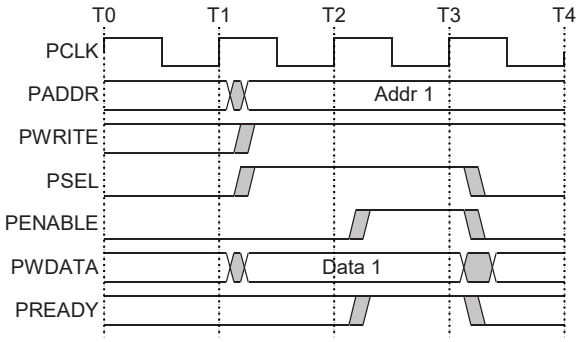

Figura 2: Transferência de escrita sem estados de espera

Em T1, uma transferência de escrita começa com os sinais PADDR, PWDATA, PWRITE e PSEL sendo registrados na borda de subida de PCLK. Isso é chamado de Fase de *Setup* da transferência de escrita.

Em T2, o sinal PENABLE é registrado na borda de subida de PCLK.

Quando em alto, PREADY indica que o escravo pode completar a transferência na próxima borda de subida de PCLK.

Os sinais PADDR e PWDATA e os sinais de controle permanecem válidos até a transferência complete em T3, o fim da Fase de Acesso.

O sinal PENABLE vai pra zero ao fim da transferência. O sinal PSEL também vai pra zero, a menos que a transferência seja seguida imediatamente por outra transferência para o mesmo periférico.

3.2.2 Com Estados de Espera

Pode-se obervar na Figura 3 como o periférico escravo pode usar o sinal PREADY para extender a transferência. Durante a fase de acesso, quando PENABLE for alto, o escravo extende a transferência tornando o sinal PREADY baixo. Os sinais a seguir permanecem invariáveis enquanto PREADY for baixo:

- PADDR
- PWRITE
- PSEL
- PENABLE
- PWDATA

PREADY pode ter qualquer valor quando PENABLE for baixo. Isso garante que os periféricos que tem um acesso fixo de dois ciclos podem fixar PREADY em alto.

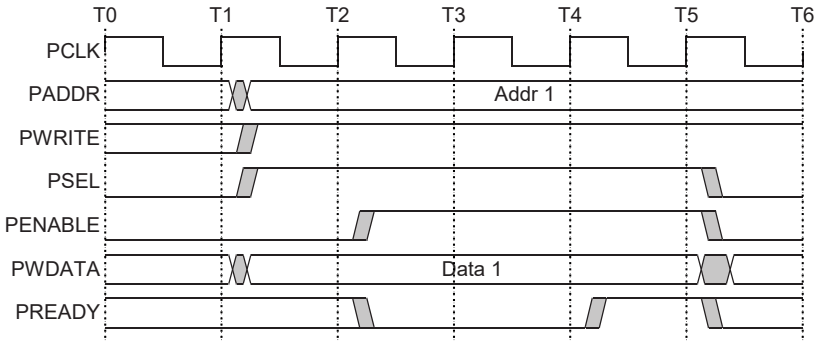

Figura 3: Transferência de escrita com estados de espera

3.3 Transferências de Leitura

Esta seção descreve os seguintes tipos de transferência de leitura:

- Sem estados de espera
- Com estados de espera

3.3.1 Sem Estados de Espera

Observe na Figura 4 a transferência de leitura. A temporização dos sinais de endereço, escrita, seleção e habilitação são descritos como na Seção 3.2.1. O escravo deve fornecer os dados antes do término da transferência de leitura.

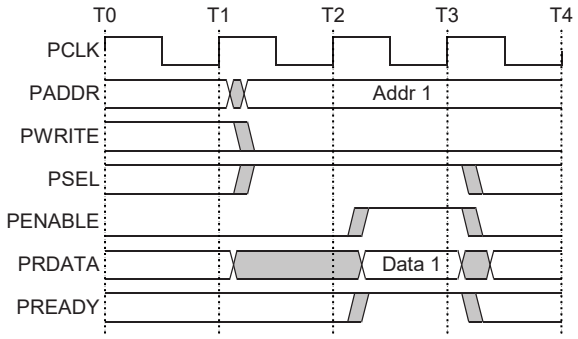

Figura 4: Transferência de leitura sem estados de espera

3.3.2 Com Estados de Espera

Pode-se observar na Figura 5 como o PREADY pode extender a transferência. A transferência é extendida se PREADY é posto em nível baixo durante a Fase de Acesso. O protocolo garante que o que vier em seguida permanece inalterado para os ciclos adicionais:

- PADDR
- PWRITE
- PSEL
- PENABLE

Na Figura 5 vê-se que dois ciclos são adicionados usando o PREADY. Porém, pode-se adicionar qualquer número de ciclos adicionais maior que zero.

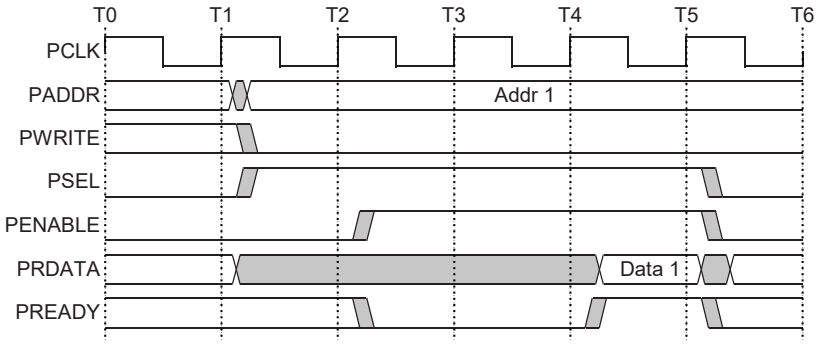

Figura 5: Transferência de leitura com estados de espera

3.4 Estados Operantes

Observa-se na Figura 6 a atividade operacional do AMBA APB.

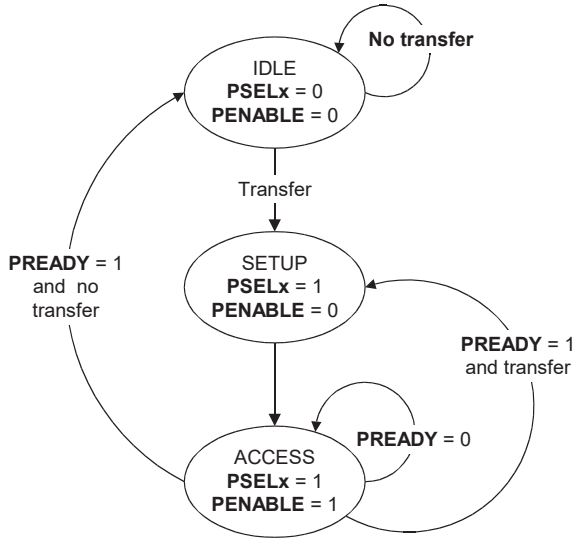

Figura 6: Diagrama de estados

A máquina de estados opera ao longo dos estados a seguir:

IDLE — Estado *default* do APB

SETUP — Quando uma transferência é requesitada, o barramento muda para o estado de SETUP, onde o sinal de seleção apropriado, PSELx, é setado. O barramento somente permanece no estado SETUP para um ciclo de relógio e sempre move para o estado de ACCESS na próxima borda de subida do relógio.

ACCESS — O sinal de habilitação, PENABLE, é setado no estado de ACCESS. Os sinais de endereço, escrita, seleção e dados de escrita devem permanecer estáveis durante a transação do estado de SETUP para o de ACCESS.
A saída do estado ACCESS é controlado pelo sinal PREADY do escravo:

- Se PREADY é mantido baixo pelo escravo, então o barramento do periférico permanece no estado de ACCESS.
- Se PREADY é setado alto pelo escravo, então é saído do estado de ACCESS e o barramento retorna ao estado IDLE se nenhuma outra transferência for requisitada. Alternativamente, o barramento desloca diretamente ao estado SETUP se outra transferêcia seguir-se.

4 Especificações

Pretende-se desenvolver um IP (Intelectual Property) de um MD5, sem pré-processamento e que use interface AMBA APB. O diagrama de bloco do MD5 pode ser visto na Figura 7 e as descrições das entradas e saídas na Tabela 3.

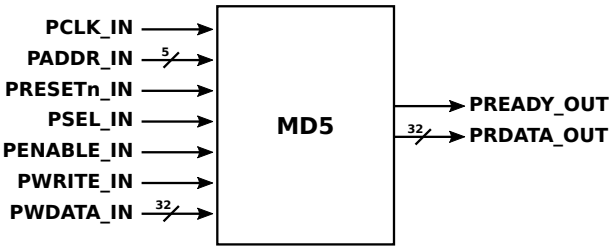

Figura 7: Digrama de bloco do MD5

Tabela 3: Entradas e saídas do MD5

Pino	Tipo	Largura (*bits*)	Descrição
PCLK_IN	Entrada	1	Entrada do sinal de relógio de $50\,MHz$
PADDR_IN	Entrada	5	Entrada do barramento de endereço. Seu valor pode variar de $0x00$ a $0x13$ para acessar um registrador válido
PRESETn_IN	Entrada	1	Sinal do tipo nível baixo ativo usado para reiniciar os registradores e o sistema
PSEL_IN	Entrada	1	Sinal usado pelo protocolo AMBA APB selecionando o periférico escravo (MD5)
PENABLE_IN	Entrada	1	Sinal que habilita a transferência APB
PWRITE_IN	Entrada	1	Quando em ”1”, indica ao periférico escravo que uma escrita será feita. Quando em ”0”, uma leitura será feita
PWDATA_IN	Entrada	32	Entrada do barramento de dados
PREADY_OUT	Saída	1	Saída de sinal usado pelo periférico escravo para extender uma transferência de dados
PRDATA_OUT	Saída	32	Saída do barramento de dados

O conjunto de registradores e seus endereços é mostrado na Tabela 4.

Tabela 4: Mapa de registradores do SHA-1

Nome	Tipo	Largura (*bits*)	Endereço (hexa)	Descrição
Reg_In_Msg [15:0]	Escrita	32	$0x00$ - $0x0f$	Dezesseis registradores que armazenarão os 512 *bits* do bloco pré-processado.
Reg_Out_MD [3:0]	Leitura	32	$0x10$ - $0x13$	Quatro registradores que armazenarão a mensagem sumária
Reg_Out_Default	Escrita/ Leitura	32	$0x14$ - $0x1f$	Registrador usado em caso do valor de endereço não condizer com um endereço válido. O valor desse registrador é nulo.

A mensagem recebida pelo bloco deverá estar devidamente pré-processada. Assim, será enviado o bloco de 512 *bits* pré-processado que será computado para produzir a mensagem sumária. A frequência do relógio usada para o desenvolvimento do bloco será de $50\,MHz$.

Antes do envio da mensagem, deve-se resetar o sistema para garantir os estados iniciais dos registradores internos que armazenam as constantes.

Os 512 *bits* do bloco da mensagem serão quebrados em dezesseis partes e enviados de um em um, sendo armazenados em Reg_In_Msg [15:0]. Ao receber a última parte do bloco de mensagem, o bloco deve

usar da extensão do estados de espera e só sair ao concluir o cálculo do *hash*, podendo receber um próximo bloco para a continuação do cálculo. O resultado do cálculo do *hash* deve ser armazenado no registrador Reg_Out_MD [3:0].

Em caso de ser endereçado um valor dentro da faixa $0x14$ - $0x1f$ (ou seja, que não apontaria pra nenhum registrador válido), o sistema esreverá/lerá em/de um registrador nomeado Reg_Out_Default.