GTA / UFRJ
GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO

Politécnica UFRJ

COPPE UFRJ

CPE716
Internet das Coisas

Aula 2

cruz@gta.ufrj.br http://gta.ufrj.br/~cruz

PEE – Programa de Engenharia Elétrica
Universidade Federal do Rio de Janeiro

Nesta apresentação

- Objetos inteligentes
  - Princípios
  - Arquitetura básica
    - Módulos
    - Comunicação entre módulos
  - Prototipagem

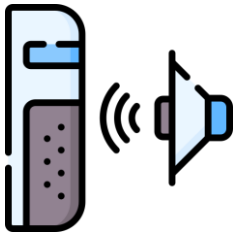
Bibliografia

- Esta apresentação utiliza informações das seguintes fontes:
  - Bibliografia do curso
    - Hanes, Salgueiro, Grossetete, Barton e Henry. "IoT Fundamentals", Ciscopress, 2011.
  - Documentação Intelbras
    - https://backend.intelbras.com/sites/default/files/2022-05/mibo-cam-rtsp.pdf
  - Documentação Espressif ESP32
    - https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
    - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ulp.html
  - Vídeos do canal do fabricante Microchip
    - https://www.youtube.com/watch?v=qTLRRg6Mee0 (I2C)
    - https://www.youtube.com/watch?v=Lozf9sceW_o (DRAM)
    - Https://www.youtube.com/watch?v=kU2SsUUsftA (SRAM)

Objetos inteligentes

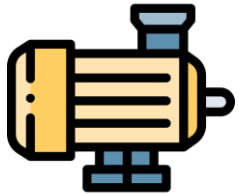
Sensores

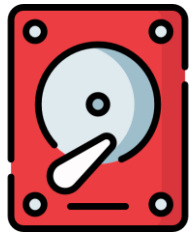
- Obtém informação do mundo externo
  - Temperatura
  - Humidade
  - Imagem
  - Código de barras
  - Presença
  - Luminosidade



Atuadores

- Alteram algo no mundo externo
  - Motor
  - Tela
  - Led
  - Alto-falante
  - Fechadura
  - Relé





Objetos inteligentes

- Coletar informações do ambiente → Sensoreamento
- Enviar informações coletadas → Comunicação
- Tomar decisão → Processamento
- Atuar sobre ambiente → Atuação

Características

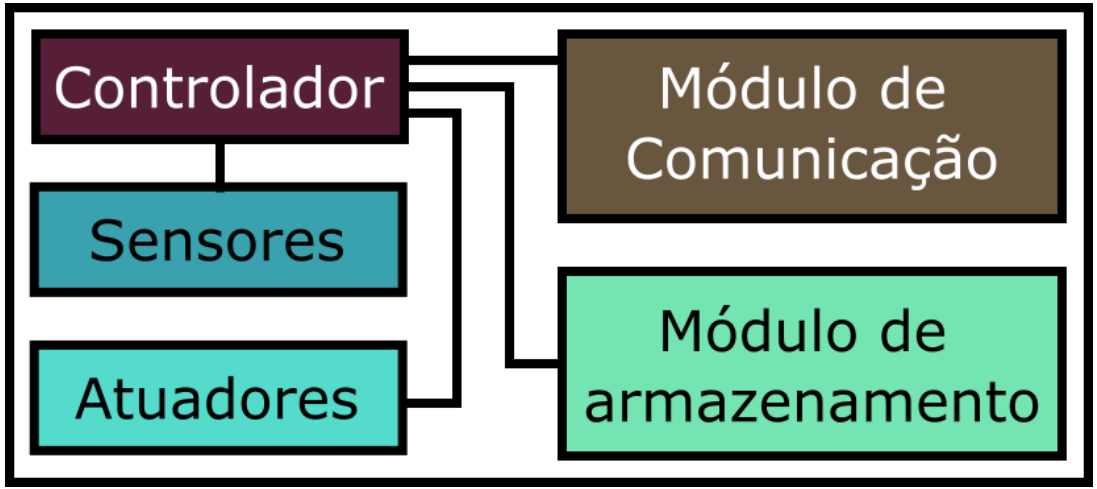
- Custo baixo
- Mobilidade
- Proximidade com os eventos de interesse
- Poder de processamento limitado
- Capacidade de comunicação limitada
- Capacidade energética limitada

Arquitetura das coisas

Arquitetura

- Módulos
  - Controlador (CPU)
  - Módulo de armazenamento
  - Módulo(s) de comunicação
  - Sensor(es)
  - Atuador(es)

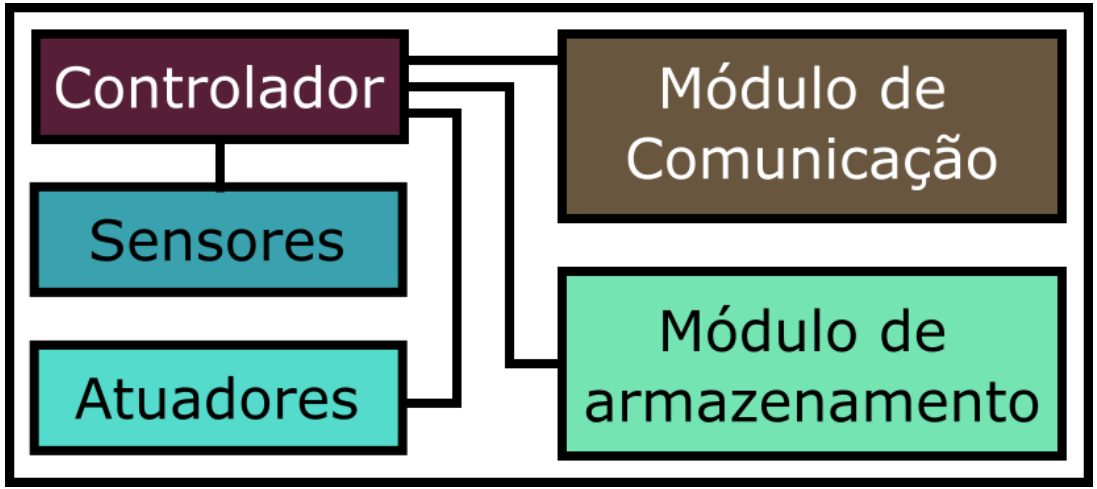
- Interfaces
  - Todos se comunicam com e através do controlador



Arquitetura

- Módulos
  - Controlador (CPU)
  - Módulo de armazenamento
  - Módulo(s) de comunicação
  - Sensor(es)
  - Atuador(es)

- Interfaces
  - Todos se comunicam com e através do controlador



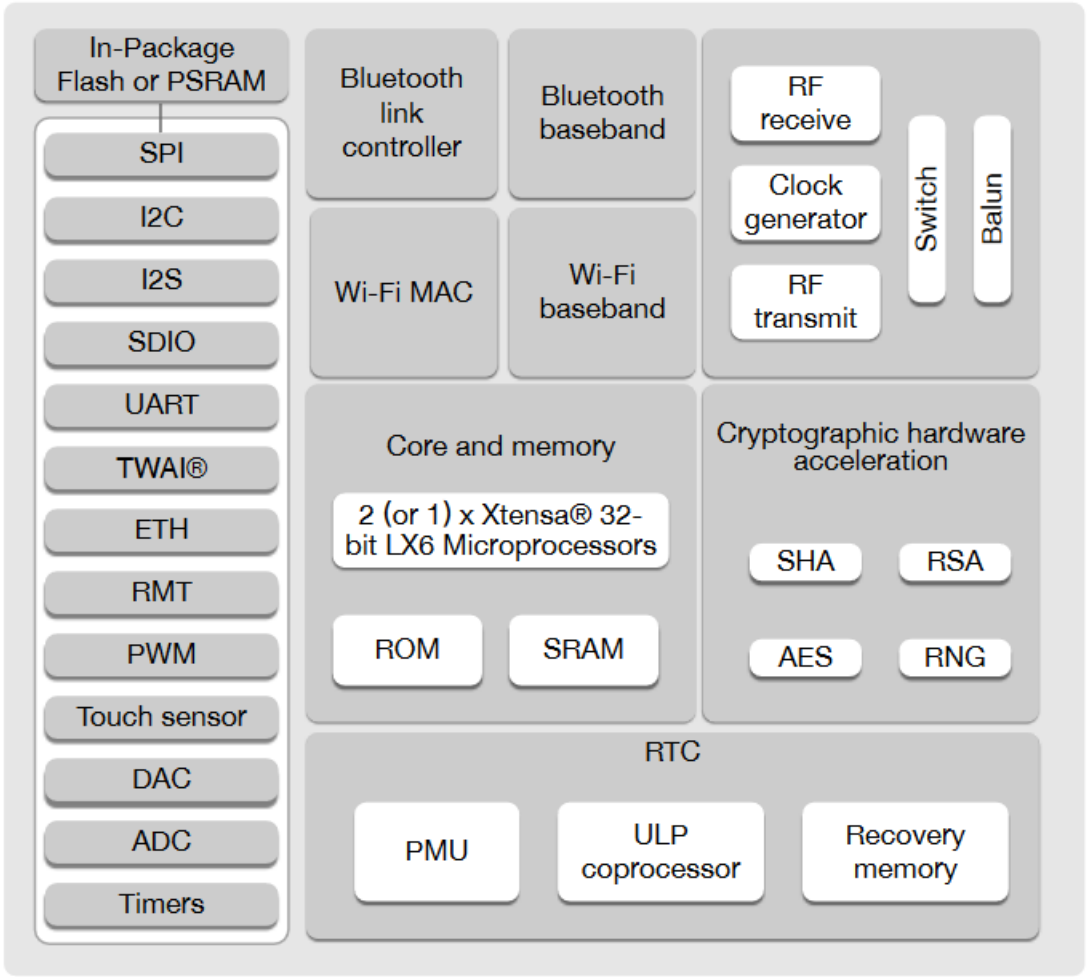
Frequentemente Módulo de Comunicação e de Armazenamento estão integrados ao Controlador

Controlador – System on a Chip (SOC)

- Módulo único de computação
  - CPU
  - RAM
  - Armazenamento
  - I/O
  - Comunicação
    - Opcional

Exemplo: ESP32

Retirado de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf



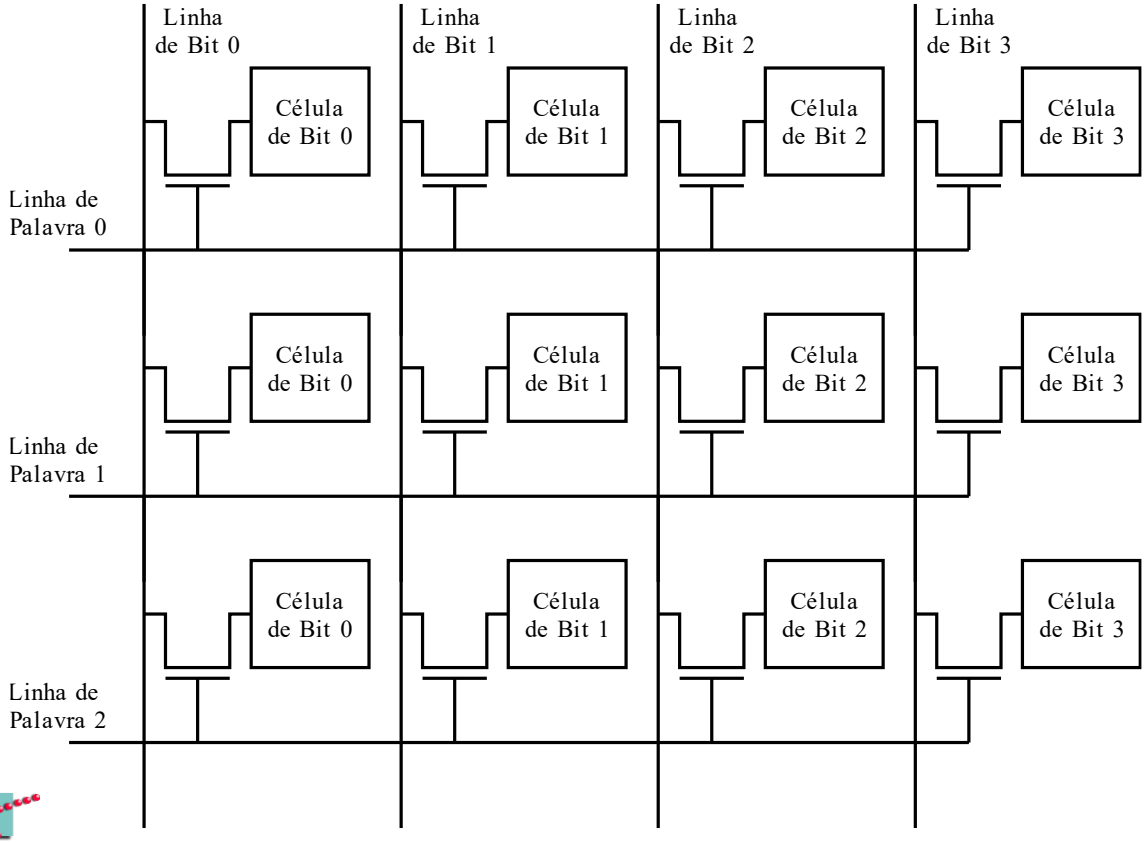
Memória (S/D/PS) RAM

Célula de bit (*bit cell*)

- Armazena exatamente um bit
  - Ou seja, unidade atômica de armazenamento
- Fica ativa para leitura/escrita quando sua palavra está ativa
- É lida/escrita junto com sua palavra
- Pode ser acessada de maneira aleatória (Random Access Memory)
- Pode ser feita de diversas maneiras diferentes

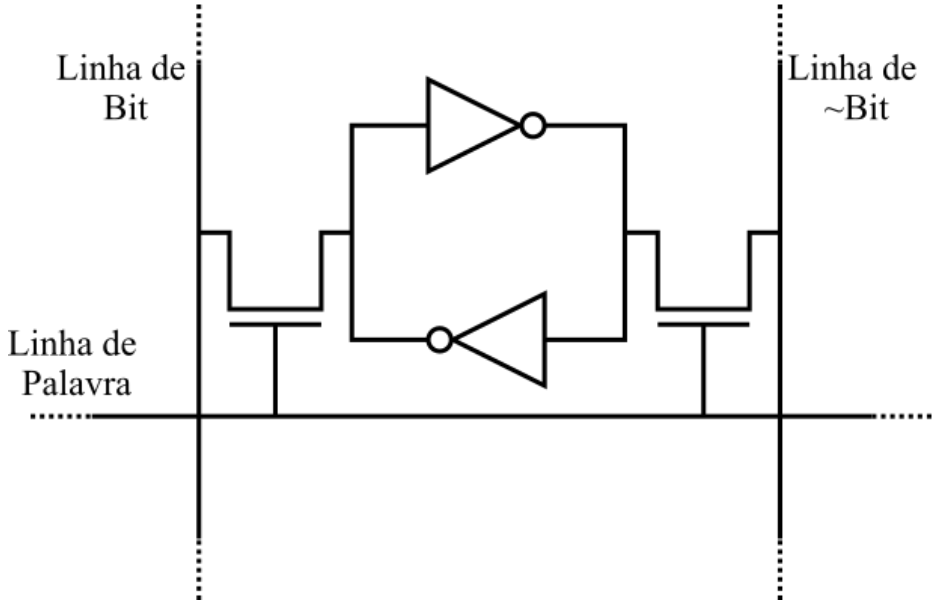
Endereçamento

- Linha de palavra
  - Ativa em um mintermo possível das entradas de endereçamento
  - Seleciona todos os bits daquela palavra
- Linha de bit
  - Recebe um bit
    - Leitura/escrita
  - Pode ser passagem para várias palavras diferentes
    - Ativas em momentos diferentes



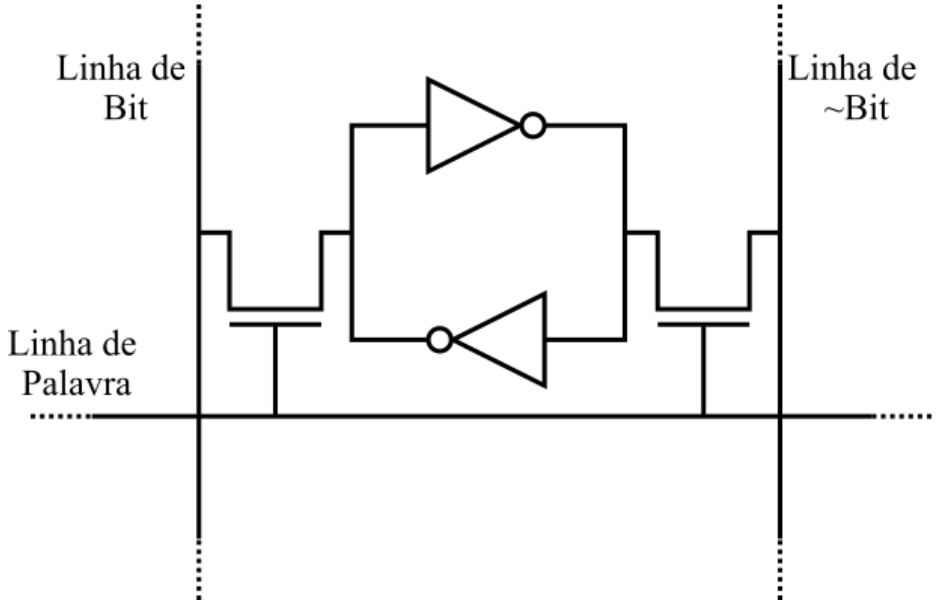
Célula SRAM (Static RAM)

- Rápidas
- Caras
- Densidade alta (6 transistores/bit)



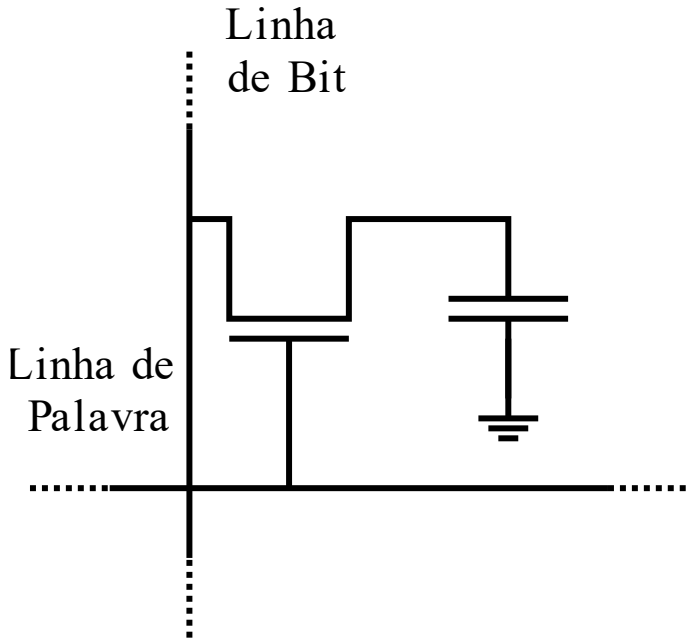
Célula SRAM

- Portas NOT em realimentação
  - Valor armazenado é estático
- Escrita feita com valor e seu complemento
- Leitura feita sem destruição do valor



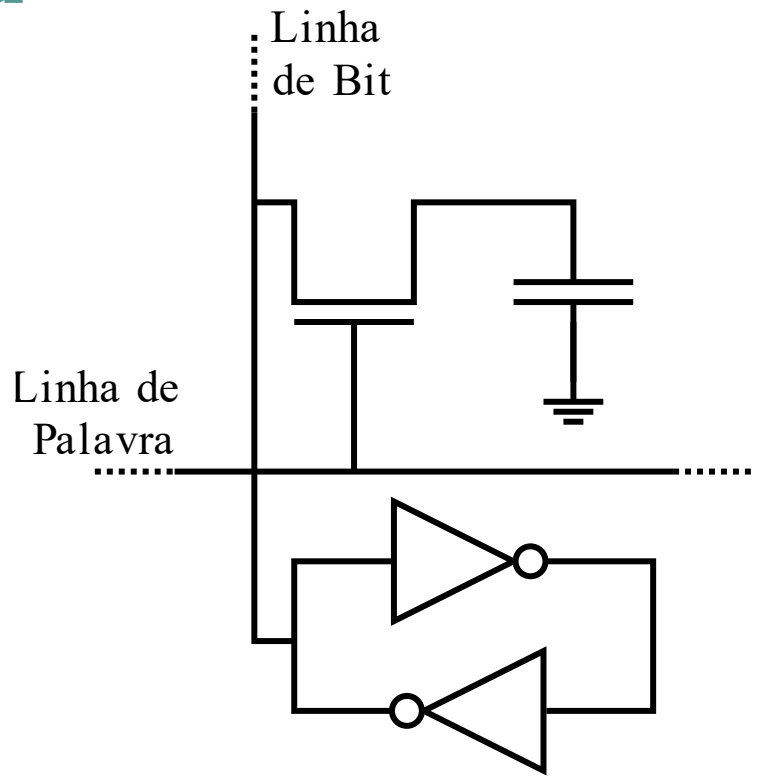
Célula DRAM

- Capacitor controlado por um transistor
  - Descarga do capacitor causa perda do valor armazenado
    - Memória “dinâmica”



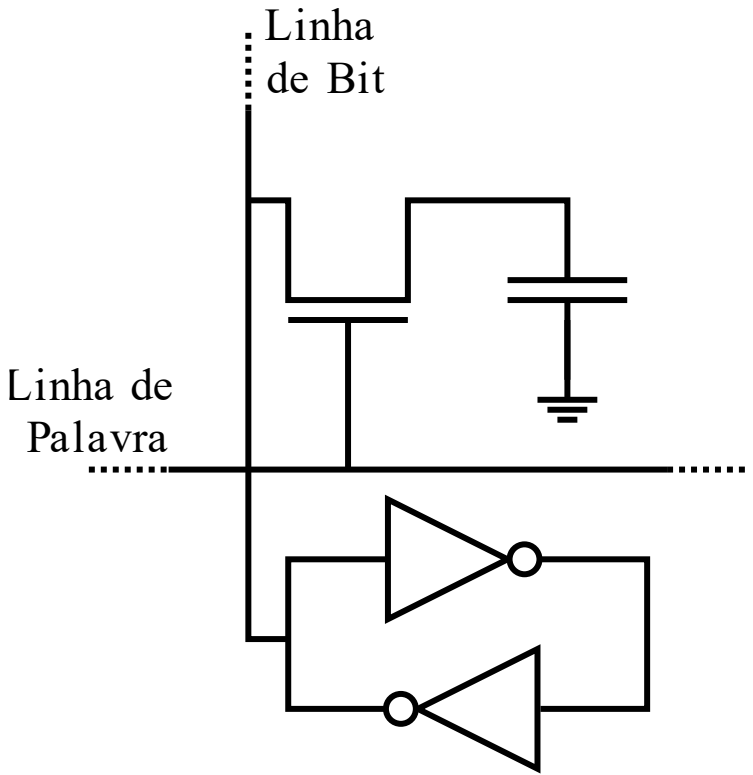
Célula DRAM: leitura

- Colocar nas portas ½ Vcc
- Ativar o transistor
  - Valor “fraquinho” do capacitor vai ativar realimentação nas portas
  - Portas vão reforçar leitura
  - Portas vão recarregar capacitor



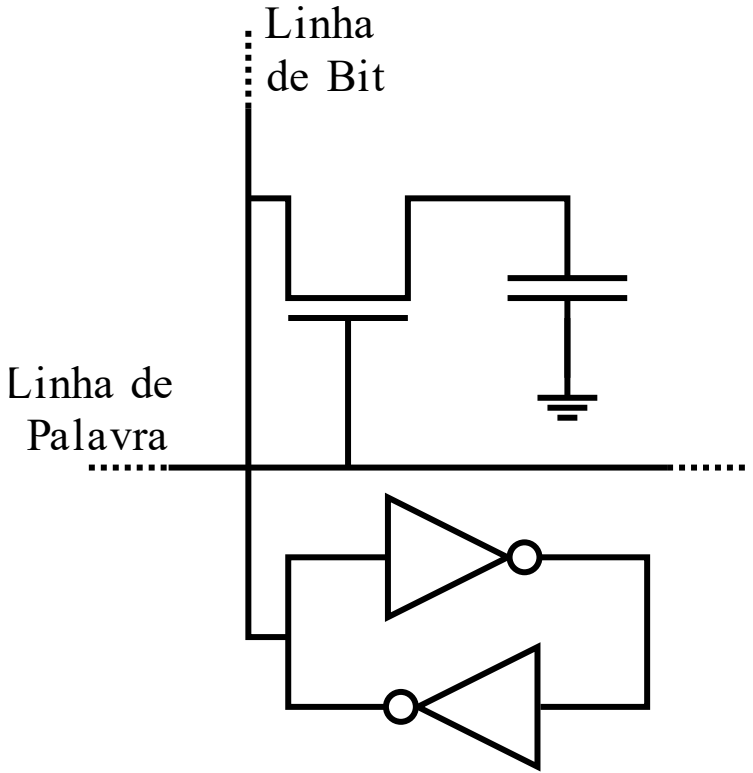
Célula DRAM: carga

- Realizar uma leitura de cada célula
- Nas memórias DDR4, deve ser realizada a cada 1/60 de segundo



Memória PSRAM

- Memória DRAM
  - Encapsulada junto com recarga
- Controlador não precisa se preocupar com a recarga
- Controlador vê como um módulo externo



Código para acessar a memória PSRAM

```
#include <Arduino.h>

void setup() {
  log_d("Total heap: %d", ESP.getHeapSize());
  log_d("Free heap: %d", ESP.getFreeHeap());
  log_d("Total PSRAM: %d", ESP.getPsramSize());
  log_d("Free PSRAM: %d", ESP.getFreePsram());
}

void loop() {}
```

Testando a PSRAM
e lendo o resultado

```
psramInit(): PSRAM enabled
] setup(): Total heap: 393356
] setup(): Free heap: 367948
] setup(): Total PSRAM: 4194252
] setup(): Free PSRAM: 4194252
```

```
void setup() {
  logMemory();
  byte* psdRamBuffer = (byte*)ps_malloc(500000);
  logMemory();
  free(psdRamBuffer);
  logMemory();
}
```

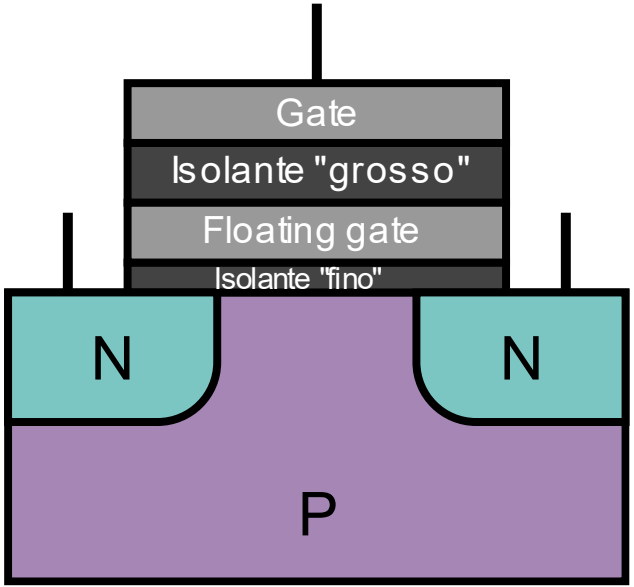
Alocando PSRAM

Código retirado de https://thingpulse.com/esp32-how-to-use-psram/

Memória Flash

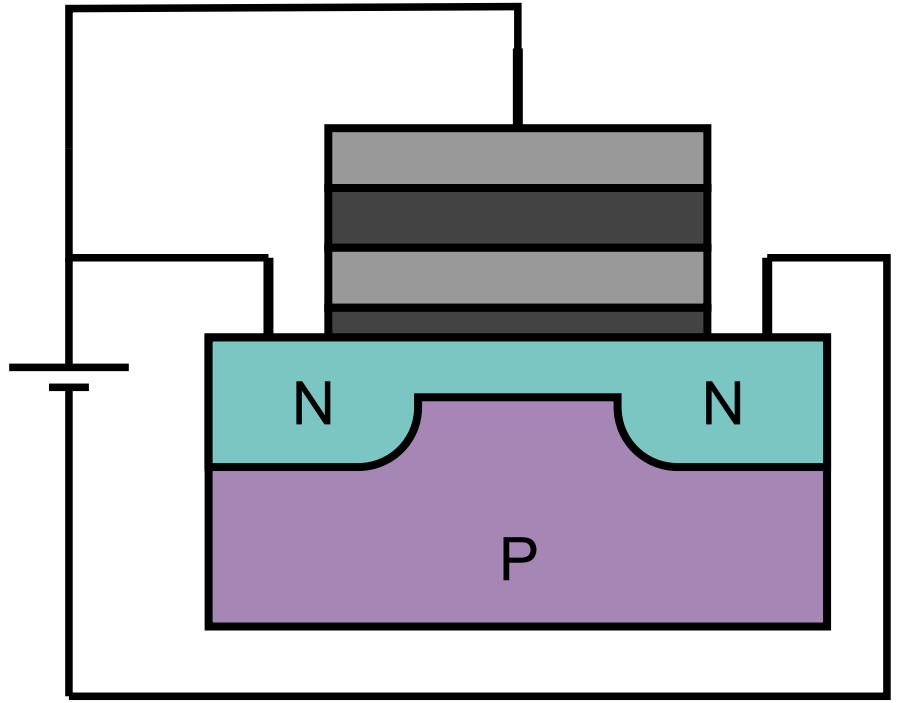
*Floating Gate Transistor**

- Gate tem uma placa condutora separada por dois isolantes
  - Um é mais fino que outro



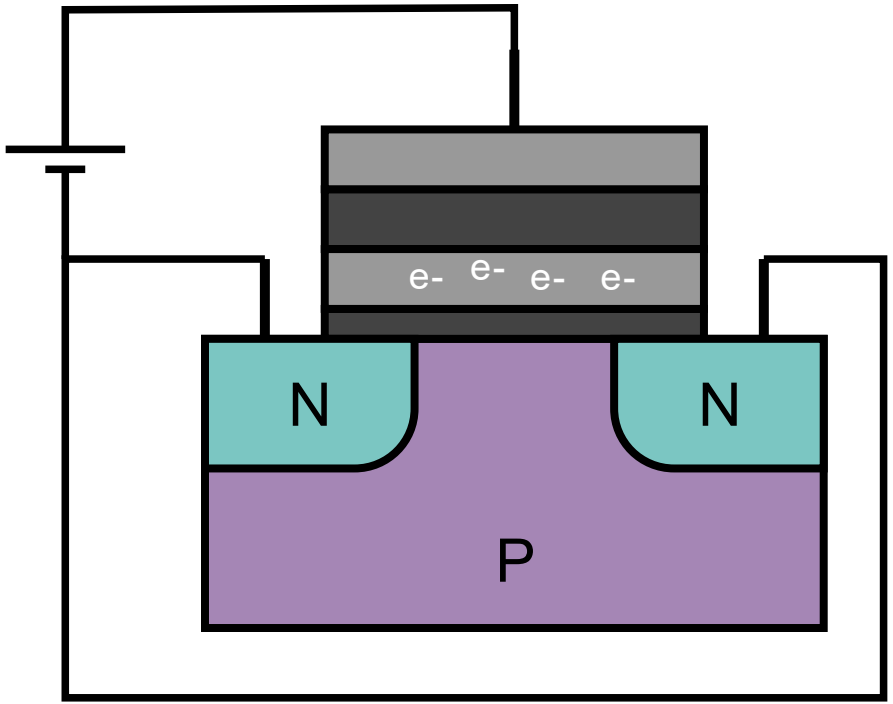
Estado inicial: valor lógico 1

- Transistor está em 1
- Pequena tensão no *gate* fecha o transistor
  - Deixa passar corrente



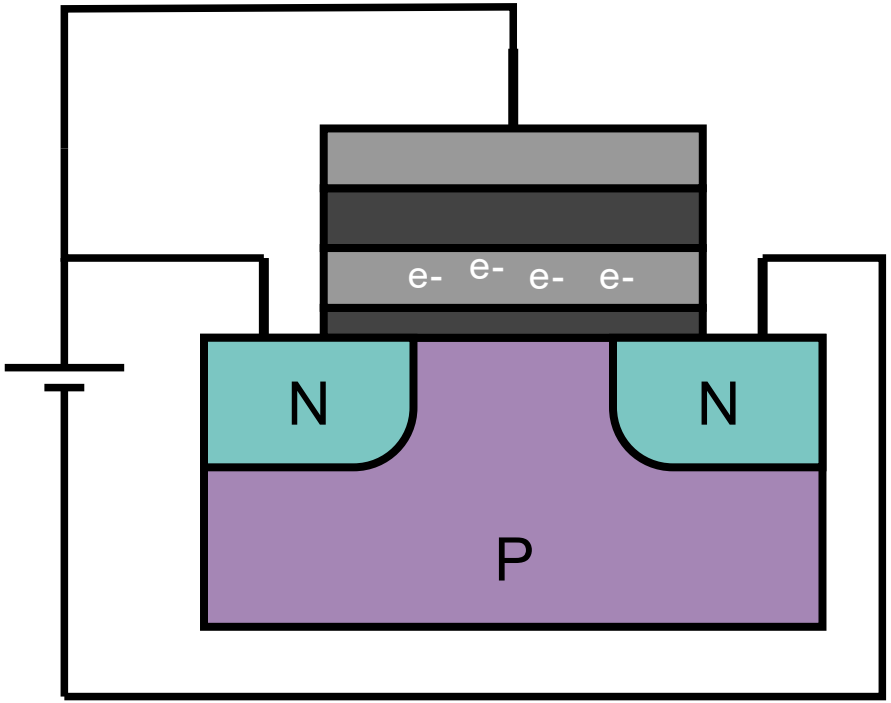
Escrita: *hot electron injection*

- Se 0:
  - Alta tensão é aplicada entre *gate* e o resto do transistor
  - Elétrons “pulam” do substrato e ficam presos no *floating gate*
    - Agora, campo elétrico torna mais difícil o transistor deixar passar corrente
- Se 1:
  - Nada



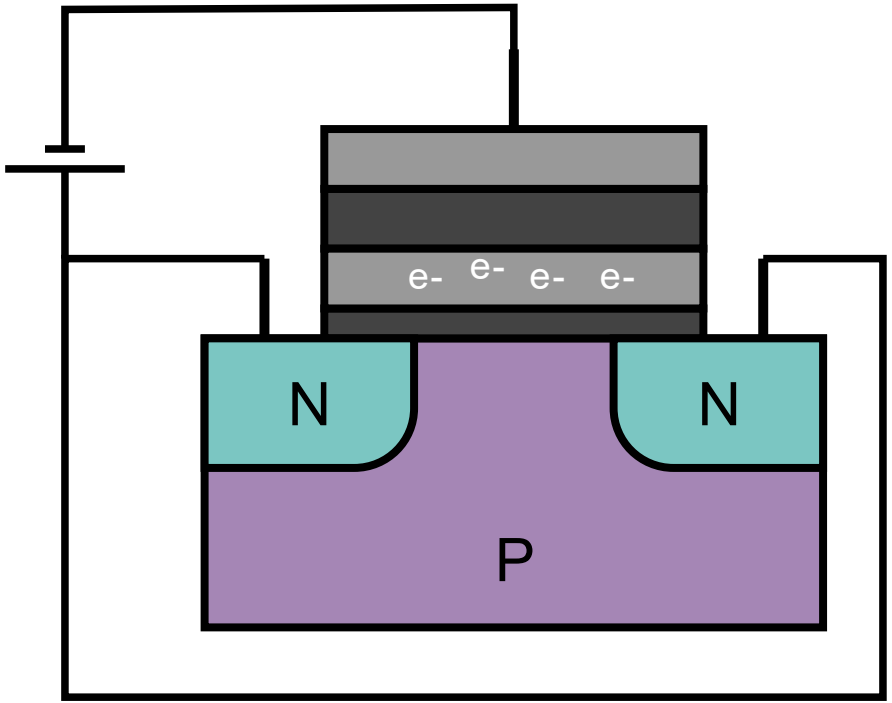
Estado final: valor lógico 0

- Transistor está em 0
- Pequena tensão no *gate* não fecha o transistor
  - Elétrons não deixam
  - Corrente não passa



Apagamento: reverter tensão alta

- Alta tensão reversa é aplicada entre *gate* e o resto do transistor
  - Elétrons saem do *floating gate*
  - Transistor volta ao estado inicial
- Deve ser feito com todos os transistores de um mesmo *bloco*



Dados e código

- Memória dividida entre código e dados
  - Segurança
    - Entrada maliciosa de dados na memória mista pode ser executada
  - Desempenho
    - Memória de dados pode ser lenta
    - Memória de programa não pode ser lenta

Considerações sobre energia

Energia

- Alimentação pode ser limitada
  - Bateria
  - Energia solar
  - Vibração
  - ....

- Dispositivo deve executar apenas funções necessárias
  - Processamento
  - Envio de mensagens
  - Recepção de mensagens
  - Leitura de sensores

Custo energético da comunicação (RF)

Módulos ativos	Consumo
Transmissão 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	240mA
Transmissão 802.11g, DSSS 1 Mbps, POUT = +19.5 dBm	190mA
Transmissão 802.11n, DSSS 1 Mbps, POUT = +19.5 dBm	180mA
Recepção 802.11b/g/n	95~100mA
Transmissão BT/BLE, POUT = 0dBm	130mA
Recepção BT/BLE	95~100mA

Custo energético da comunicação (RF)

Módulos ativos	Consumo
Transmissão 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	240mA
Transmissão 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	190mA
Transmissão 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	180mA
Recepção 802.11b/g/n	95~100mA
Transmissão BT/BLE, POUT = 0dBm	130mA
Recepção BT/BLE	95~100mA

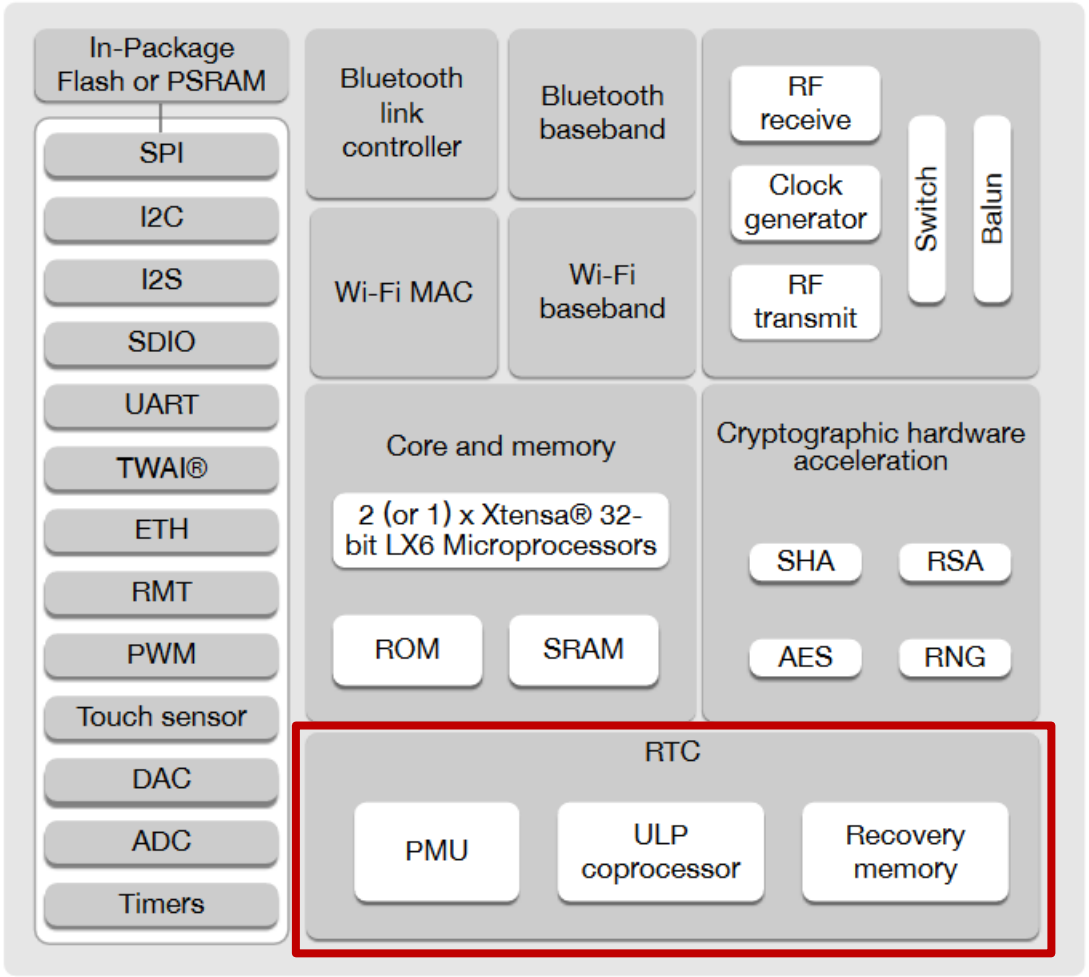
A diferença entre recepção e transmissão não é tão grande!

Módulo RTC

Real Time Clock

- Power Management Unit (PMU)
- Ultra Low Power Coprocessor
- Recovery Memory

Continuam trabalhando quando dispositivo está em *Deep Sleep*



Deep-sleep

- Desligar o processador principal

- Utilizar o coprocessador ULP (*Ultra Low Power*)

Recovery Memory

- Também chamada de RTC Memory

- Dividida em duas partes
  - *Slow RTC*
  - *Fast RTC*

Power Management Unit

- Gerencia o uso de energia

Modo	Módulos ativos	Consumo
Active	CPU + módulos de comunicação	-
Modem-sleep	CPU ligada	20mA – 44mA
Light-sleep	CPU pausada + ULP + Memória RTC + PMU	800µA
Deep-sleep	ULP + Memória RTC + PMU	10µA – 150µA
Hibernation	Apenas timer do RTC + PMU	5µA
Power off	PMU	1µA

Ultra Low Power Coprocessor

- Acessa algumas entradas
  - ADC
  - Sensor de temperatura
  - Sensores I²C

- Acessa a memória *slow* RTC

Memória RTC

Real Time Clock Random Access Memory

- Fast RTC Memory
  - Abriga o código necessário para “acordar” o processador principal
- Slow RTC Memory
  - Recebe os dados de entrada durante o Deep-sleep

Comunicação entre módulos

Comunicação entre módulos

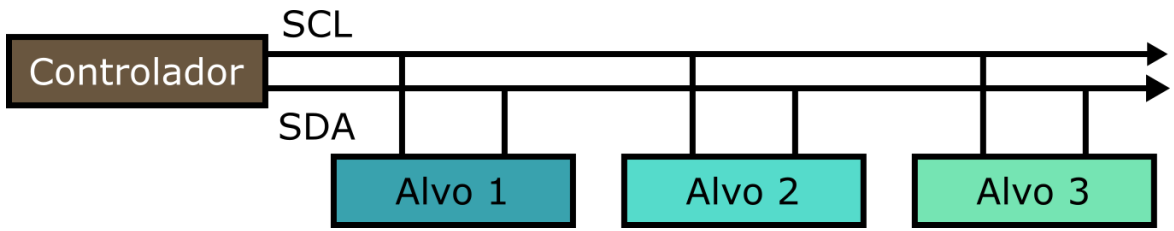
- Comunicação serial
  - Protocolos pré-definidos
    - I²C
    - SPI
    - UART
  - Protocolos decididos pela aplicação

Comunicação entre módulos

- I²C
  - Barato
    - Metade de portas lógicas pra implementar
    - 2 conexões
  - Lento
    - 1MHz
  - Half-duplex
- SPI
  - Caro
    - 3 conexões + 1 *chip select* para cada dispositivo
  - Rápido
    - 20MHz
  - Full-duplex

I²C

- Um dispositivo controlador
- Diversos dispositivos alvo
  - Até 127



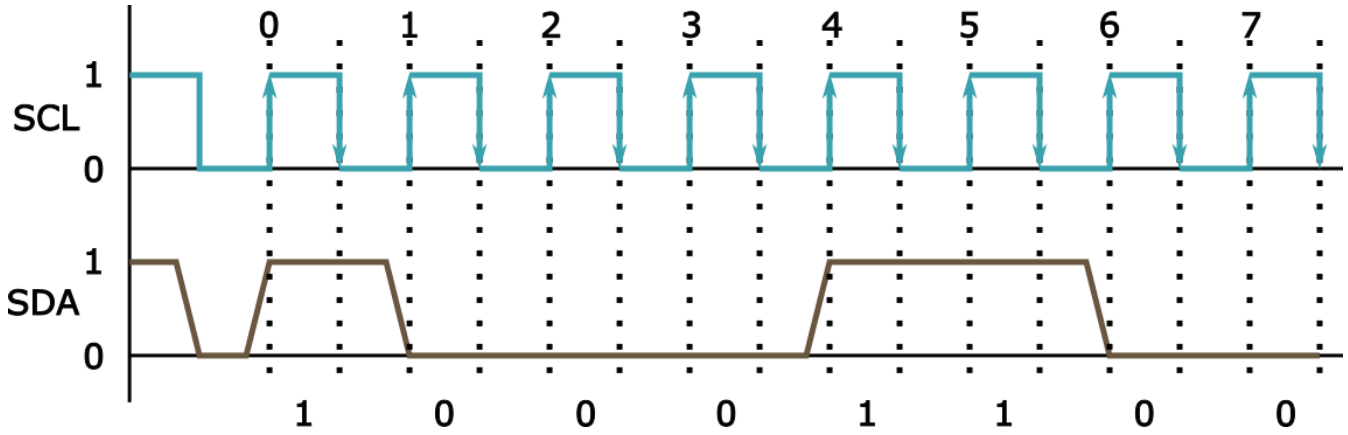
- Serial Clock Line
  - Envio de dados quando SCL = 1

- Serial Data Line
  - Coletor aberto
    - Saídas em High-Z quando ociosas ou quando nível lógico 1
    - Resistores de pull-up (omitidos)

- Dispositivos endereçados com endereço fixo de 7 bits
  - Alguns fabricantes usam mais bits

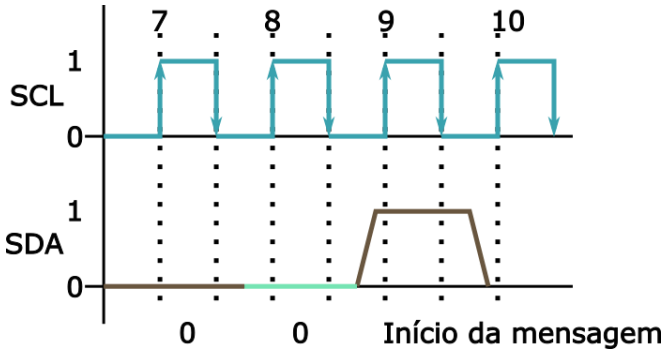
Inicialização

- Coordenador inicia a comunicação com a condição START nas linhas
  - Linhas SDA e SCL ficam em 0
  - Clock é iniciado em SCL
- Todos os alvos escutarão o barramento, esperando endereço
- Coordenador envia, bit a bit, o endereço desejado pela linha SDA mais bit $\mathrm{R}/\overline{W}$
- Alvo que possuir o endereço desejado irá realizar a ação seguinte



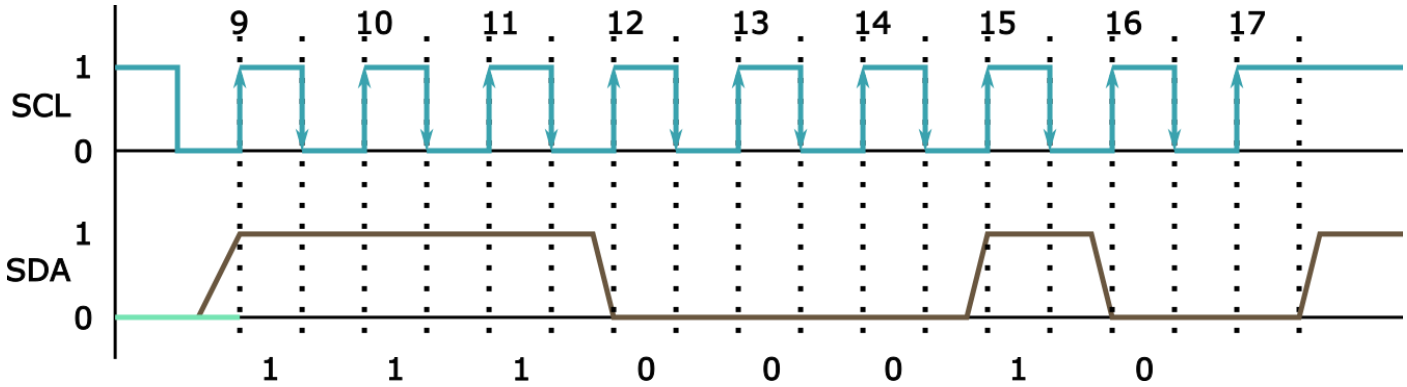
Reconhecimento do endereço

- Alvo deve enviar reconhecimento (Acknowledgement – ACK)
  - Alvo coloca SDA em 0 até o próximo pulso de CLK
  - Demais dispositivos não fazem nada
- Coordenador inicia o clock em SCL ao detectar SDA em 0
- Alvo libera linha SDA no primeiro bit 1 detectado em SCL
- Barramento estará livre para o Coordenador



Envio de dados

- Coordenador envia um Byte e depois um STOP (SCL = SDA = 1)



Comunicação Alvo-Coordenador

- Coordenador inicia comunicação
  - Mesmo procedimento
- Coordenador envia endereço
- Alvo endereçado responde com os dados na linha SDA
- Clock é gerado pelo Coordenador em SCL
- Alvo manda cada bit quando SCL for 1

Disputa pelo barramento

- Barramento pode ter mais de um Coordenador
- Quando um Coordenador escuta um START, espera um STOP para tentar enviar dados pelo barramento
  - Evita colisões
- Um coordenador pode não escutar um START
  - P.ex. foi reiniciado durante uma transmissão
  - É necessário um protocolo de controle de congestionamento

Controle de congestionamento

- Quando um Coordenador coloca nível alto na linha, sempre verifica se esse 1 foi efetivado
  - Colocar nível alto significa “não fazer nada”, já que dispositivos estão em coletor aberto
  - Se nível alto não aparecer na linha, significa que algum outro Coordenador colocou a linha em 0
    - Ou seja, linha está em uso
- Se Coordenador detecta que o barramento está em uso, inicia procedimento de *backoff*
- *Backoff* consiste em esperar uma condição STOP até enviar o dado

Créditos

Os ícones desta apresentação foram feitos por Freepic e retirados de www.flaticon.com.

GTA / UFRJ

GRUPO DE TELEINFORMÁTICA E AUTOMAÇÃO

www.gta.ufrj.br