CprE 288 – Introduction to Embedded Systems

Instructors:
Dr. Phillip Jones

Announcements

- HW 6: Due Tue 10/4
- Exam 1: Thurs Oct 6
  - HW 1 – HW5
- Quiz 5: Thursday, first 10 minutes of class on Canvas
  - GPIO Readings from HW 5
  - ~~Interrupt readings from HW6~~
  - CPRE 288 Datasheet trainer
  - **Mandatory one side of 1 page of notes (you will submit to Canvas as part of Class participation grade)**

BITWISE OPERATIONS

Why Bitwise Operation

Why use bitwise operations in embedded systems programming?

Each single bit may have its own meaning

- Push button array: Bit *n* is 0, if push button *n* is pushed
- LED array: Set bit *n* to 0 to light LED *n*

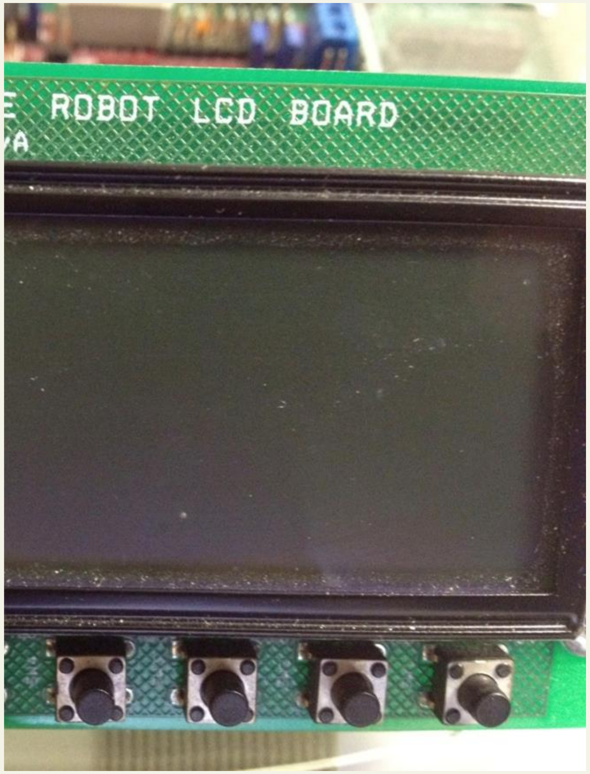
Data from/to I/O ports may be packed

- Two bits for shaft encoder, six bits for push button packed in PINC
- Keypad input: three bits for row position, three bits for column position

Data in memory may be packed to save space

- Split one byte into two 4-bit integers

Why Bitwise Operation



SW4 SW3 SW2 SW1
Bit 3 Bit 2 Bit 1 Bit 0

Read the input:

GPIO_PORTE_R;

How to have application determine which button is being pushed?

Buttons connected to PORTE, bits 3-0

Bitwise Operations

Common programming tasks:

- Clear/Reset certain bit(s)
- Set certain bit(s)
- Test if certain bit(s) are cleared/reset
- Test if certain bit(s) are set
- Toggle/invert certain bits
- Shift bits around

Bitwise Operators: Clear/Force-to-0 Bits

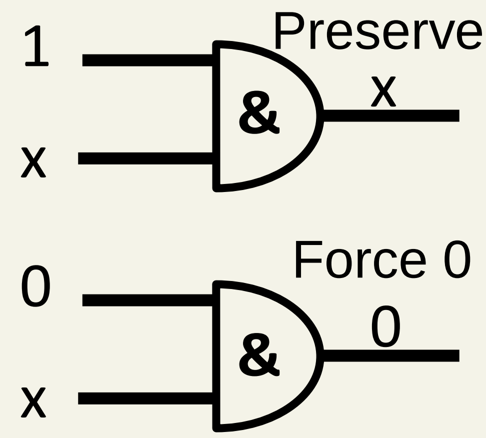
C bitwise AND: **&**

ch = ch & 0x3C;   What does it do?

Consider a single bit *x*

x AND 1 = x   Preserve

x AND 0 = 0   Clear/Force 0



Truth Table

Bit x	Mask bit	Bit x & Mask bit
x	0	0 (Forced to 0)
x	1	x (Value preserved)

Bitwise Operators: Clear/Force-to-0 Bits

ch = ch & 0x3C;

	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
AND	0	0	1	1	1	1	0	0
	0	0	x_5	x_4	x_3	x_2	0	0

Clear bits 7, 6, 1, 0

Preserve bits 5, 4, 3, 2

Clear bit(s): Bitwise-AND with a mask of 0(s)

Bitwise Operators: Clear/Reset Bits

Another example:

char op1 = 1011 1100; We want to clear bit 4 to 0.

char op2 = 1110 1111; We use op2 as a mask

char op3;

```
      1011 1100
AND   1110 1111
      ---------
      1010 1100
```

op3 = op1 & op2;

Class Exercise

char ch;

Clear every other bit of ch starting from bit-position 0

Bitwise Operators: Set Bits

C bitwise OR: |

ch = ch | 0xC3;   What does it do?

Consider a single bit *x*

- *x* OR 1 = 1   Set
- *x* OR 0 = x   Preserve

Bitwise Operators: Set Bits

```
ch = ch | 0xC3;
```

	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
OR	1	1	0	0	0	0	1	1
	1	1	x_5	x_4	x_3	x_2	1	1

Set bits 7, 6, 1, 0

Preserve bits 5, 4, 3, 2

Set bit(s): Bitwise-OR with a mask of 1(s)

Bitwise Operators: Set Bit

Another example:

char op1 = 1000 0101; We want to set bit 4 to 1.

char op2 = 0001 0000; We use op2 as a mask

char op3;

op3 = op1 | op2;

```
     1000 0101
OR   0001 0000
     ---------
     1001 0101
```

Bitwise Operators: Toggle Bits

C bitwise XOR: **^**

ch = ch ^ 0x3C; What does it do?

Consider a single bit *x*

x XOR 1 = $\overline{x}$ Toggle

x XOR 0 = x Preserve

Bitwise Operators: Toggle Bits

C bitwise XOR: ^ ch = ch ^ 0x3C;

	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
XOR	0	0	1	1	1	1	0	0
	x_7	x_6	$\overline{x_5}$	$\overline{x_4}$	$\overline{x_3}$	$\overline{x_2}$	x_1	x_0

Toggle bits 5, 4, 3, 2

Preserve bits 7, 6, 1, 0

Toggle bit(s): Bitwise-XOR with a mask of 1(s)

Bitwise Operators: Invert Bits

C bitwise invert: **~** ch = ~ch;

INV	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
	$\overline{x_7}$	$\overline{x_6}$	$\overline{x_5}$	$\overline{x_4}$	$\overline{x_3}$	$\overline{x_2}$	$\overline{x_1}$	$\overline{x_0}$

Example: ch = 0b00001111;

~ch == 0b11110000

Class Exercise

unsiged char ch;

short n;

Force the lower half of ch to `0`

Starting from bit 0 of ch, force every other bit to `1`

Force bit 15 and bit 0 of n to `0`

Toggle bits 7 and 6 of ch

Force the lower half of ch to `0`, and Toggle bit `7` of `ch`.

Bitwise Operators: Shift-Left

```
unsigned char my_reg = 0b00000001;
unsigned char shift_amount = 5;
unsigned char my_result;

my_result = my_reg << shift_amount;
```

00000**001**

00100000

<<, shifts “my_reg”, “shift_amount” places to the left

0s are shifted in from the right

Bitwise Operators: Shift-Right Logical

unsigned char my_reg = 0b10000000;

unsigned char shift_amount = 5;

unsigned char my_result;

my_result = my_reg **>>** shift_amount;

10000000

00000100

With unsigned type, **>>** is **shift-to-right logical**

0s are shifted in from the left

Bitwise Operators: Shift-Right Arithmetic

```
signed char my_reg = 0b10000000;
unsigned char shift_amount = 5;
unsigned char my_result;

my_result = my_reg >> shfit_amount;
```

10000000
11111100

```
my_reg = 0b01111111;
my_result = my_reg >> shfit_amount;
```

01111111
00000011

With signed type, >> is **shift-right arithmetic**

Sign bit value are shifted in from the left

Exercise

unsigned char ch;

Swap the upper half and lower half of ch

Bitwise Testing

Reminder: Conditionals are evaluated on the basis of zero and non-zero (i.e. Boolean).

The quantity 0x80 is non-zero and therefore TRUE.

```
if (0x02 | 0x44)
```

TRUE or FASE?

Bitwise Testing

Example

Find out if bit 7 of variable nVal is set to 1

Bit 7 = 0x80 in hex

```
if ( nVal & 0x80 )
{       …
}
```

What happens when we want to test for multiple bits?

if statement looks only for a non-zero value, a non-zero value means at least one bit is set to TRUE

Bitwise Testing: Any Bit is Set to 1?

Example

See if bit 2 or 3 is set to 1

Bits 2,3 = 0x0C in hex

```
if (nVal & 0x0C)
{
      Some code…
}
```

What happens for several values of nVal?

nVal = 0x04	bit 2 is set	Result = 0x04	TRUE
nVal = 0x0A	bits 3,1 are set	Result = 0x08	TRUE
nVal = 0x0C	bits 2,3 are set	Result = 0x0C	TRUE

Bitwise Testing: All Bits Are Set to 1?

Why does this present a problem?

What happens if we want to see if both bits 2 and 3 are set, not just to see if one of the bits is set to true?

Won't work without some other type of test

Two solutions

Test each bit individually

```
if ( (nVal & 0x08) && (nVal & 0x04))
```

Check the result of the bitwise AND

```
if ((nVal & 0x0C) == 0x0C)
```

Why do these solutions work?

1. Separate tests – Check for each bit and specify logical condition
2. Equality test – Result will only equal 0x0C if bits 2 and 3 are set

Bitwise Testing

- Testing if <u>any</u> of a set of bits is set to 1
  1) Decide which bits you want to test
  2) Isolate those bits (i.e. force all other bits to 0)
- Testing if <u>all</u> bits of a set of bits are set to 1
  1) Decide which bits you want to test
  2) Isolate those bits (i.e. force all other bits to 0)
  3) Compare for equality with the Mask
- For the case of testing for bits set to 0. Follow bit(s) set to 1 testing procedure, but invert the variable that you are testing.
- Generic systematic checking example

```
if( (x & MASK_ALL1s) == MASK_ALL1 &&
    (~x & MASK_ALL0s) == MASK_ALL0s &&
    (x & MASK_ANY1s)  &&
    (~x & MASK_ANY0s) )
```

Exercise

char ch;

Test if any of bits 7, 6, 5, 4 is set to 1

Test if all of bits 7, 6, 5, 4 are set to 1

Test if all of bits 7 and 6 are set to 1, and if bits 5 and 4 are cleared to 0

Bitwise operations: Summary

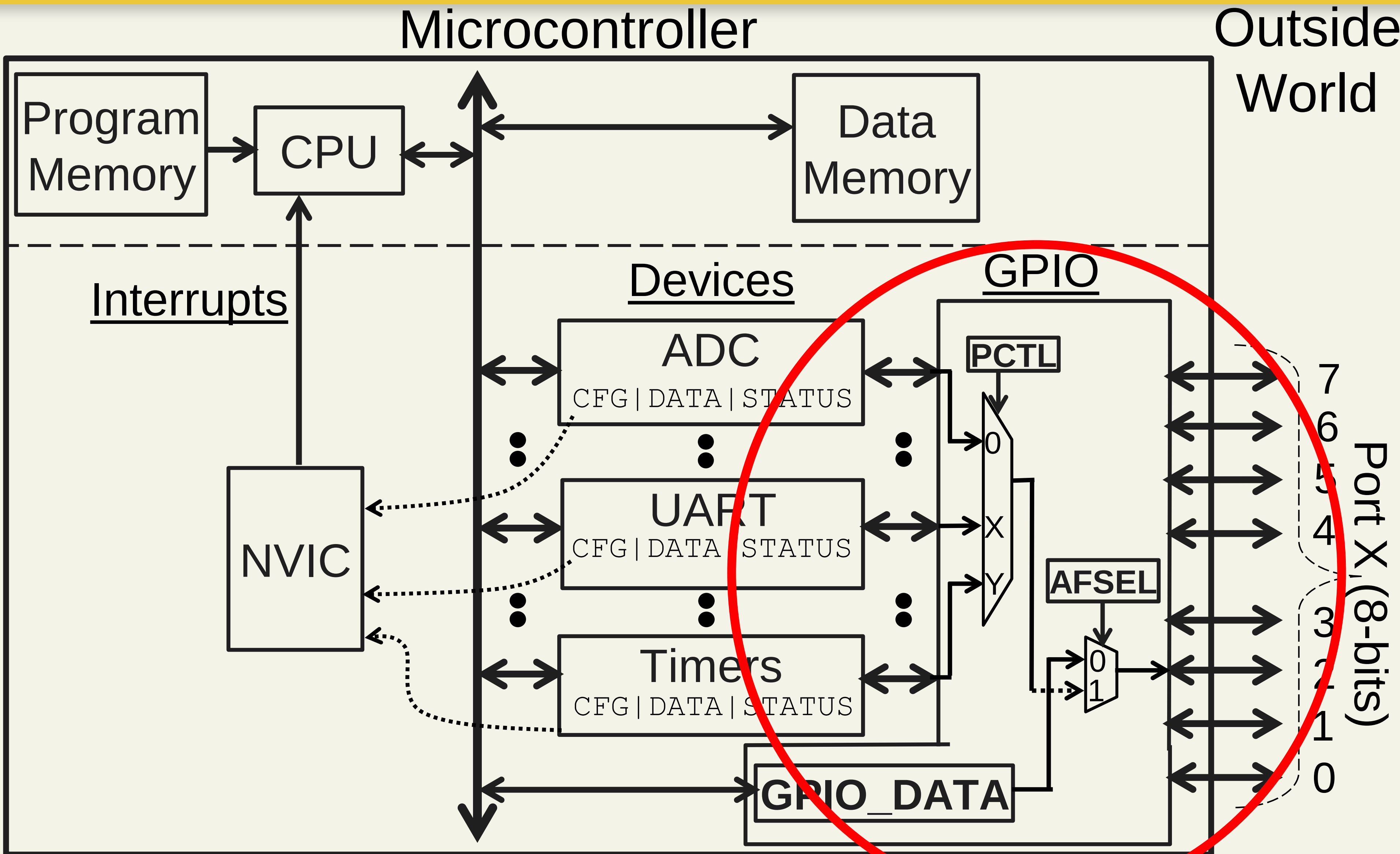
- Forcing bits to 0: & (AND)
- Forcing bits to 1: | (OR)
- Toggle bits: ^ (XOR)
- Testing for bits set to 1
- Testing for bits cleared to 0
- Generic systematic testing approach:

```
if( (val & MASK_ALL1s) == MASK_ALL1 &&
    (~val & MASK_ALL0s) == MASK_ALL0s &&
    (val & MASK_ANY1s)  &&
    (~val & MASK_ANY0s) )
```

***Where: MASK_XXX is a mask with 1s in the positions being tested**

Memory Mapped I/O

Microcontroller / System-on-Chip (SoC)



Memory Mapped I/O

- Package Pins
- Pins with shared (multiplexed) functionality
  - General Purpose I/O
  - Device I/O
- Memory mapped registers
  - Configuration registers
  - Data registers
  - Status registers

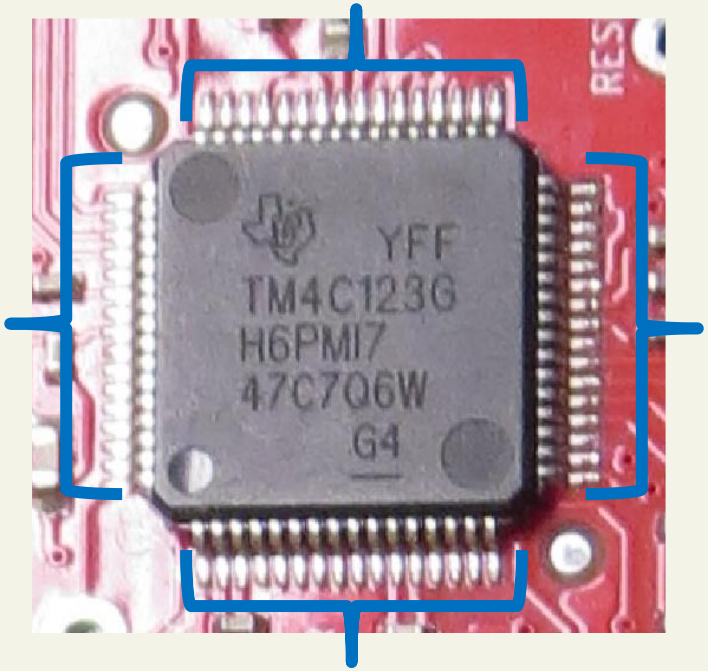
TM4C123 I/O Ports

- 64 package pins

TM4C123 I/O Ports

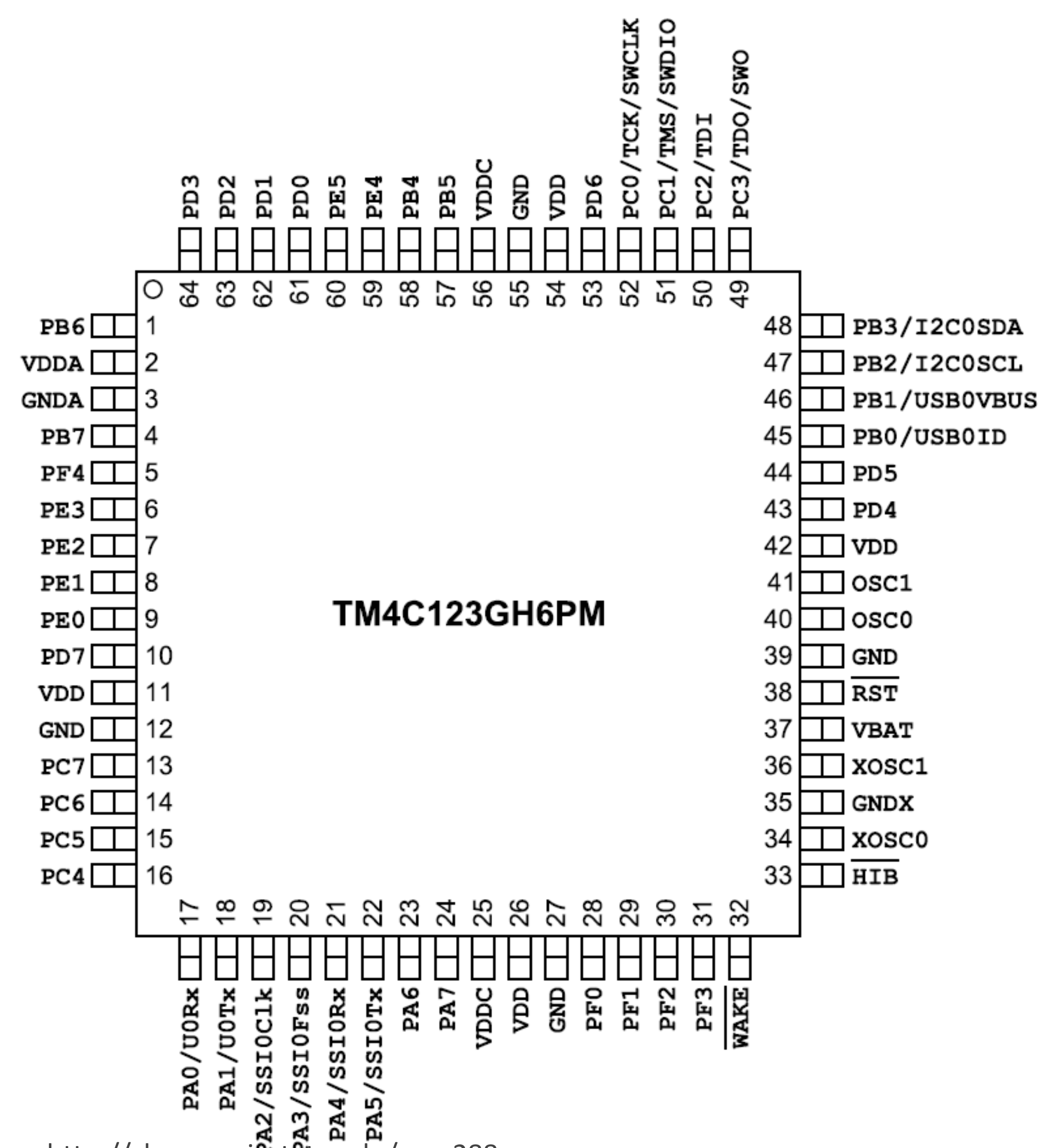
- 64 package pins

Allows Software to access the world outside of the chip



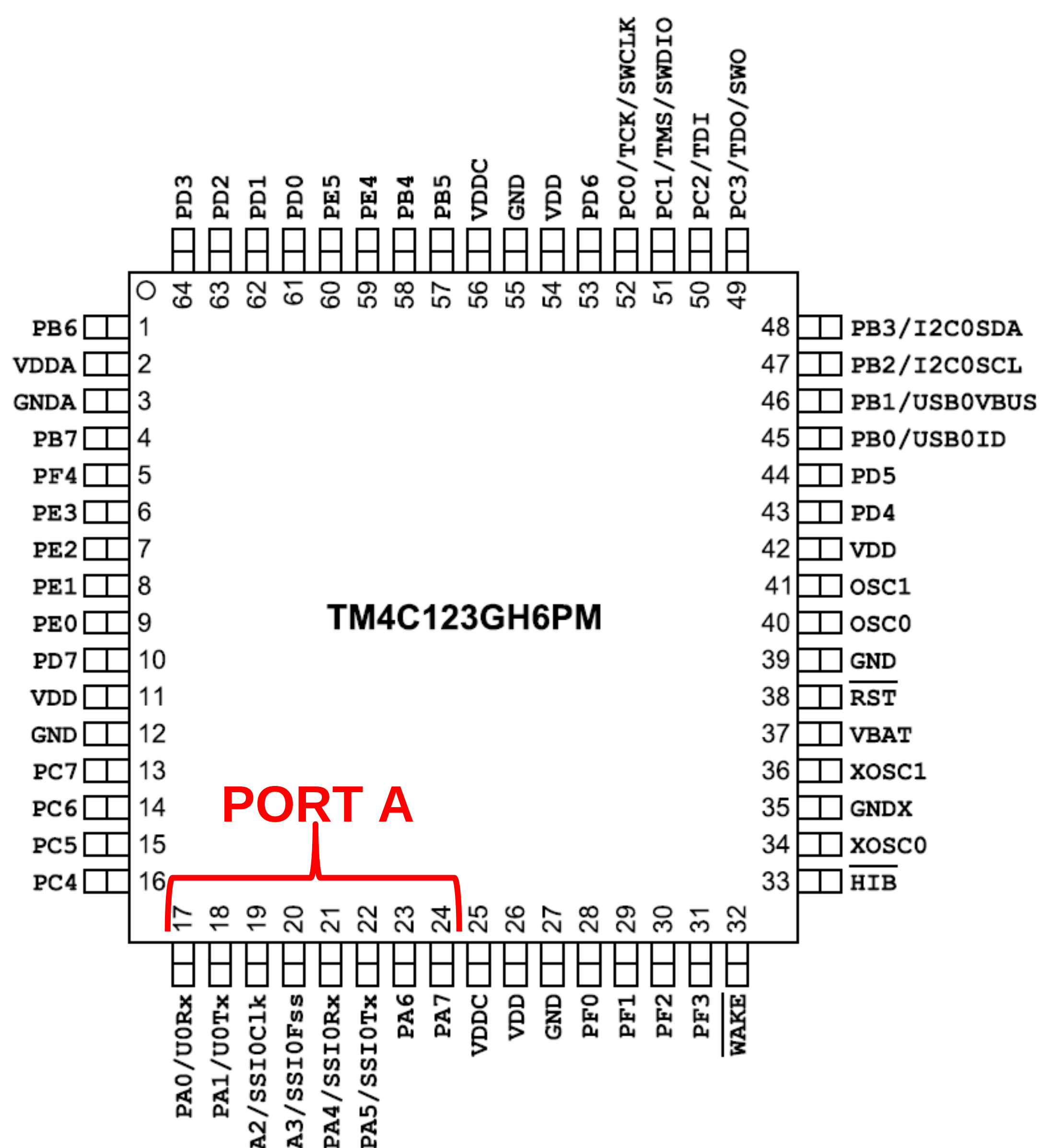
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



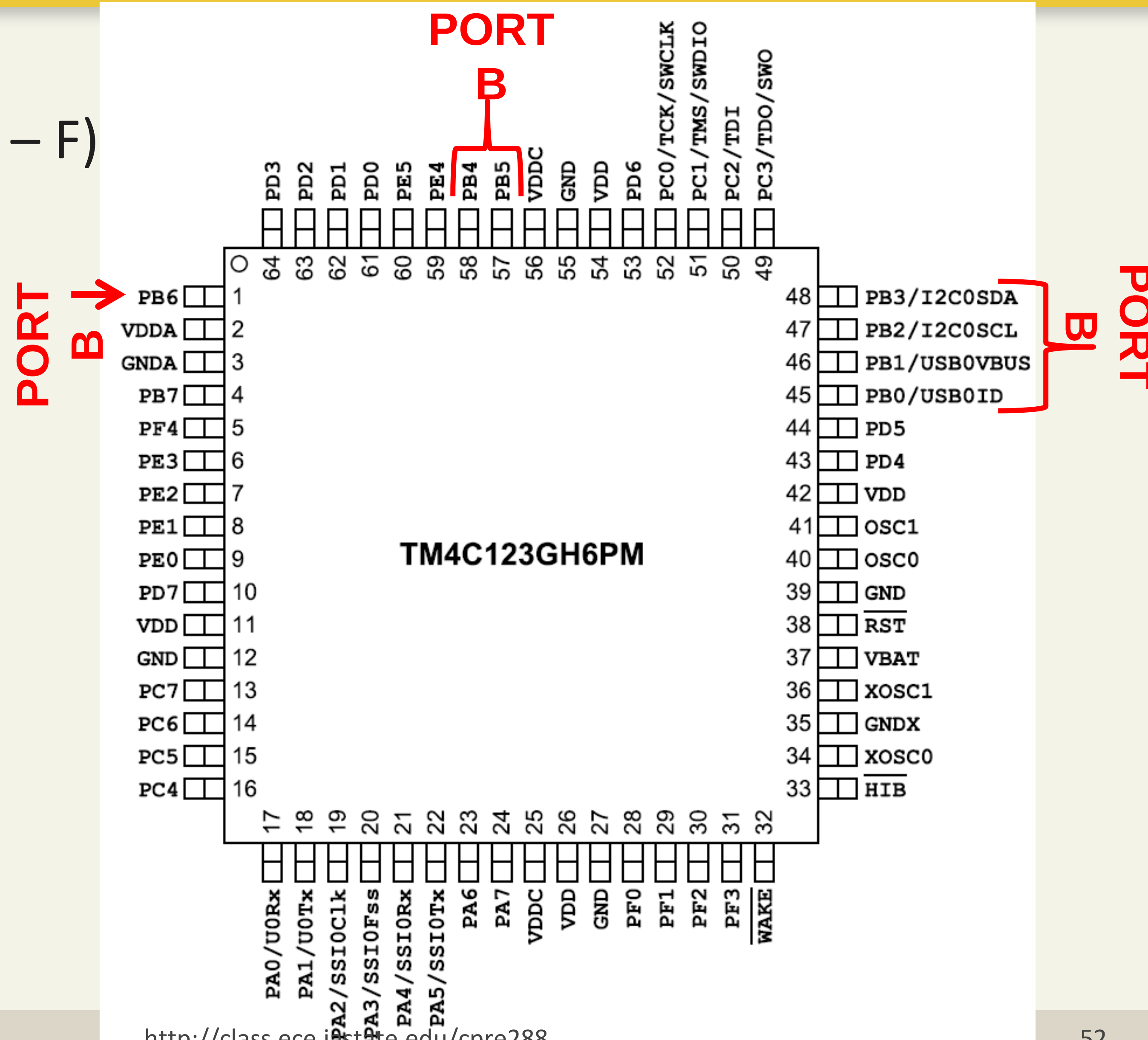
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



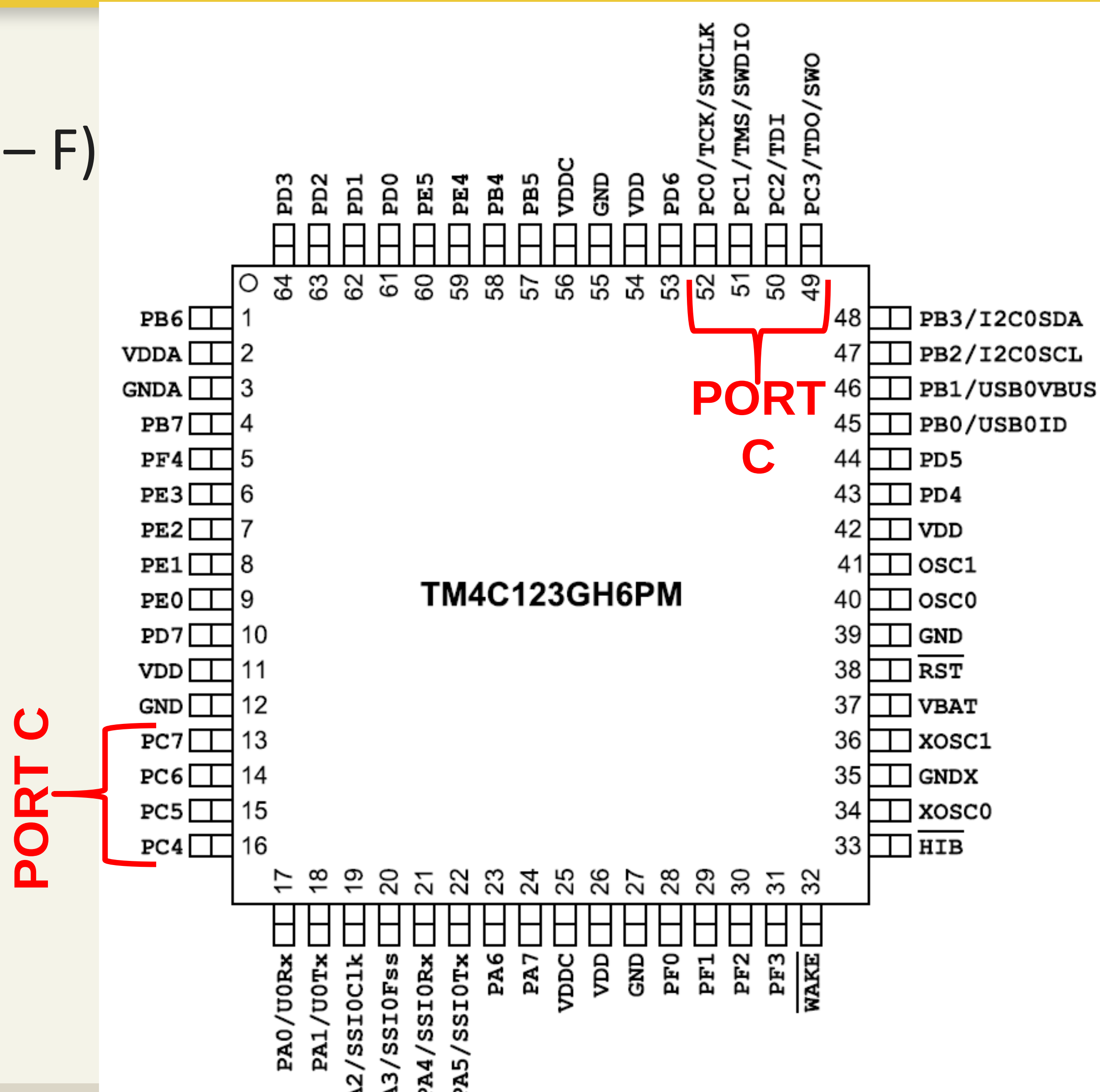
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



TM4C123 I/O Ports

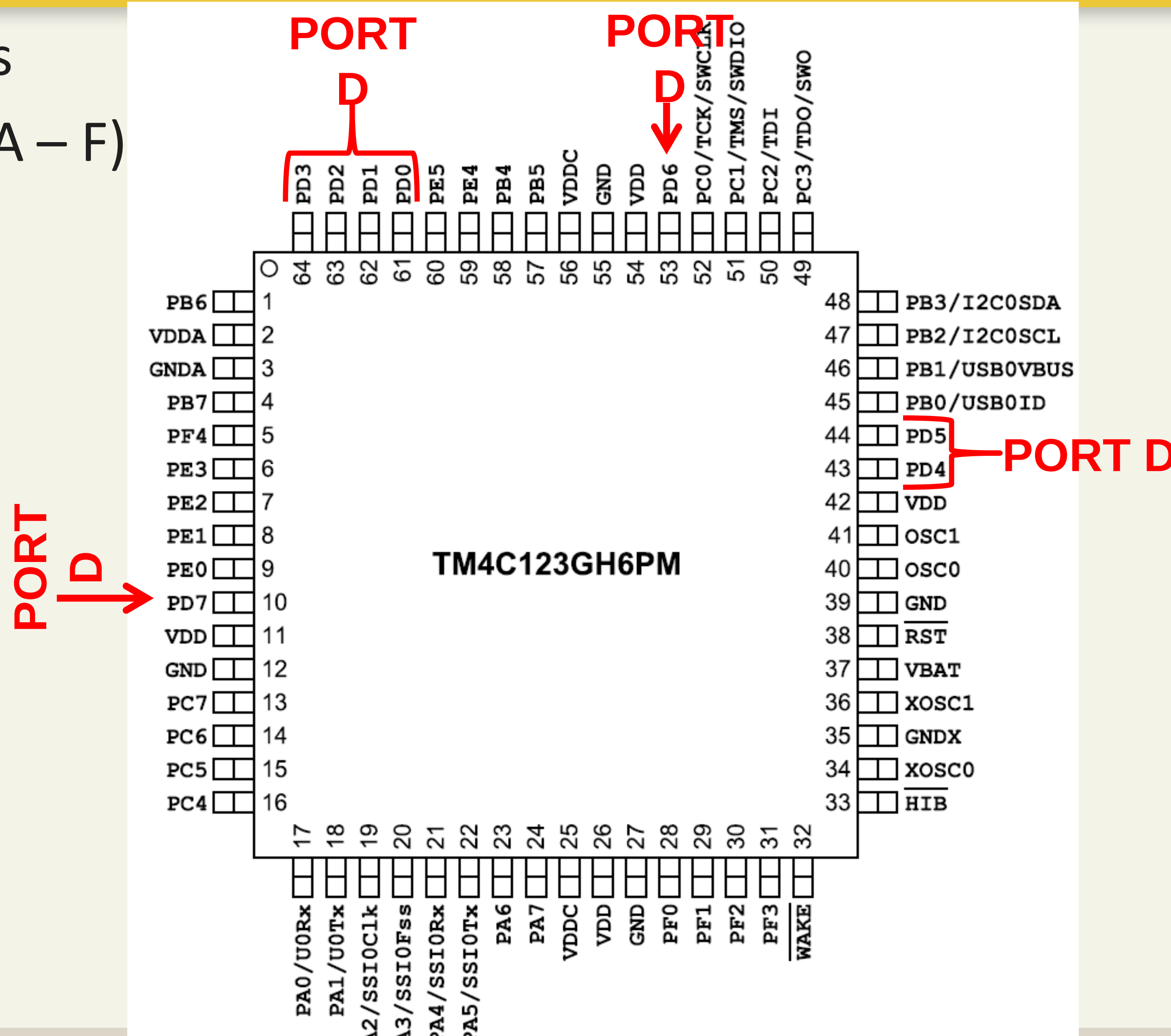
- 64 package pins
- 6 GPIOs Ports(A – F)
  - Each 8-bits



http://class.ece.iastate.edu/cpre288

TM4C123 I/O Ports

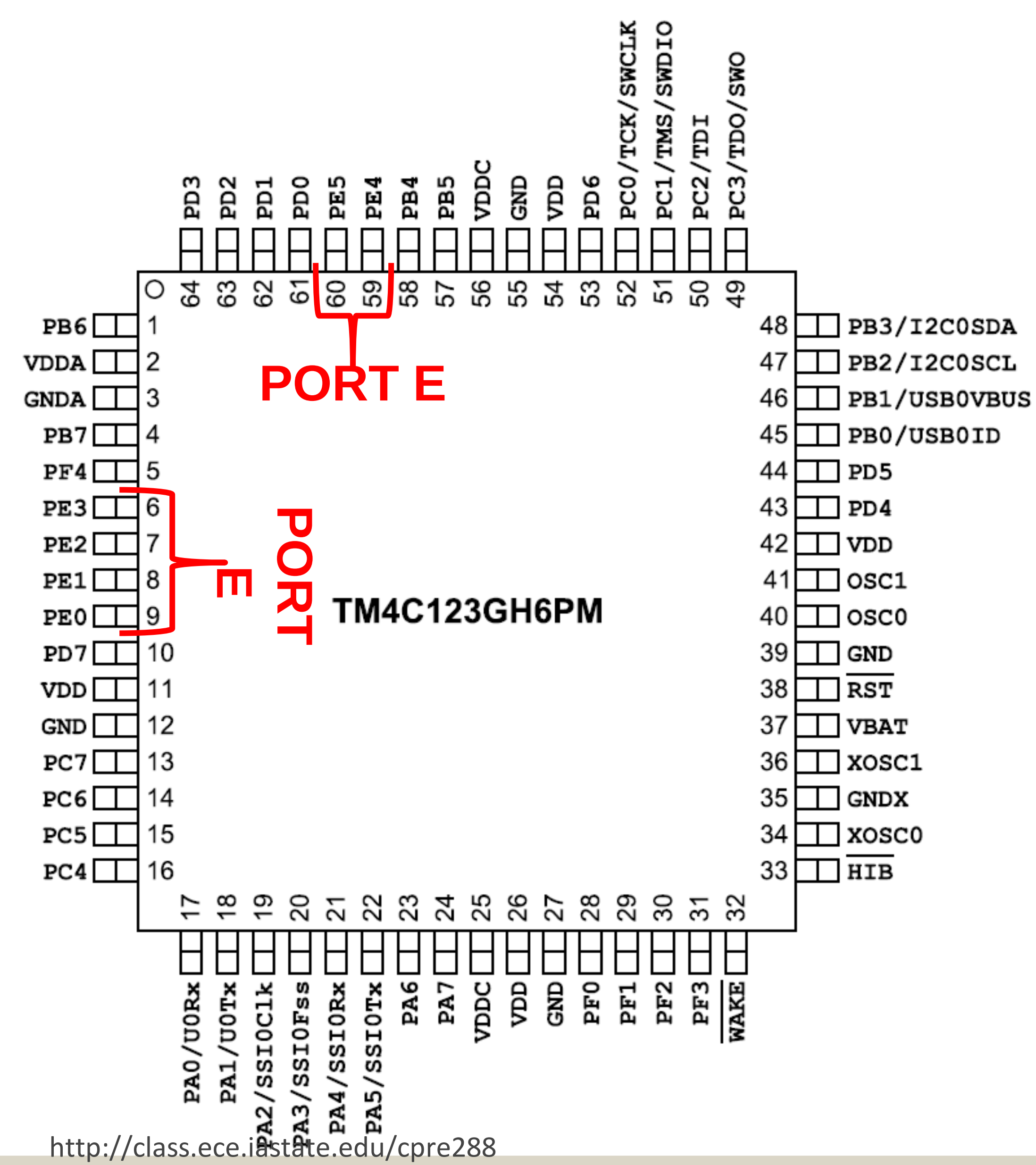
- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



http://class.ece.iastate.edu/cpre288

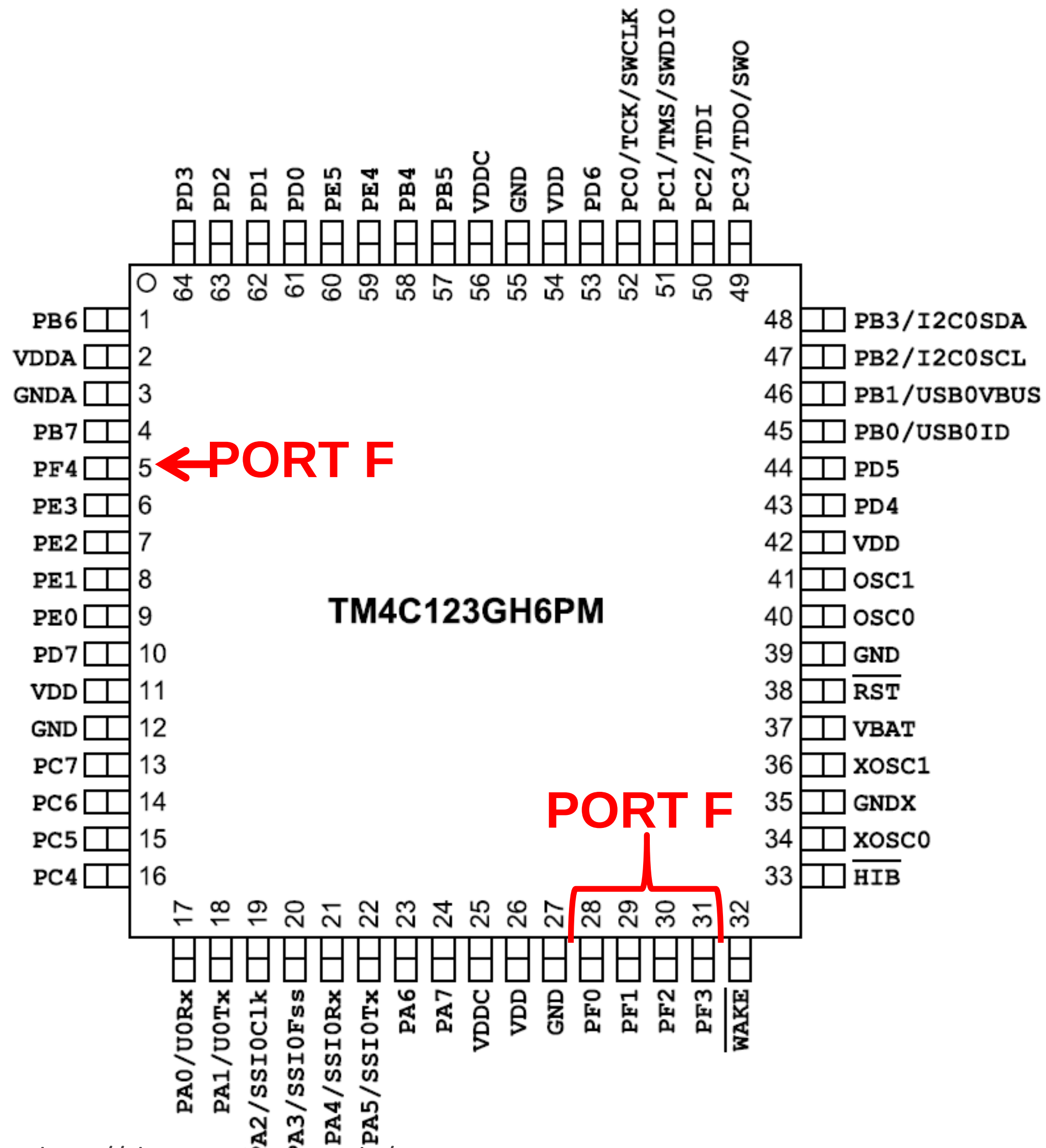
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



TM4C123 I/O Ports

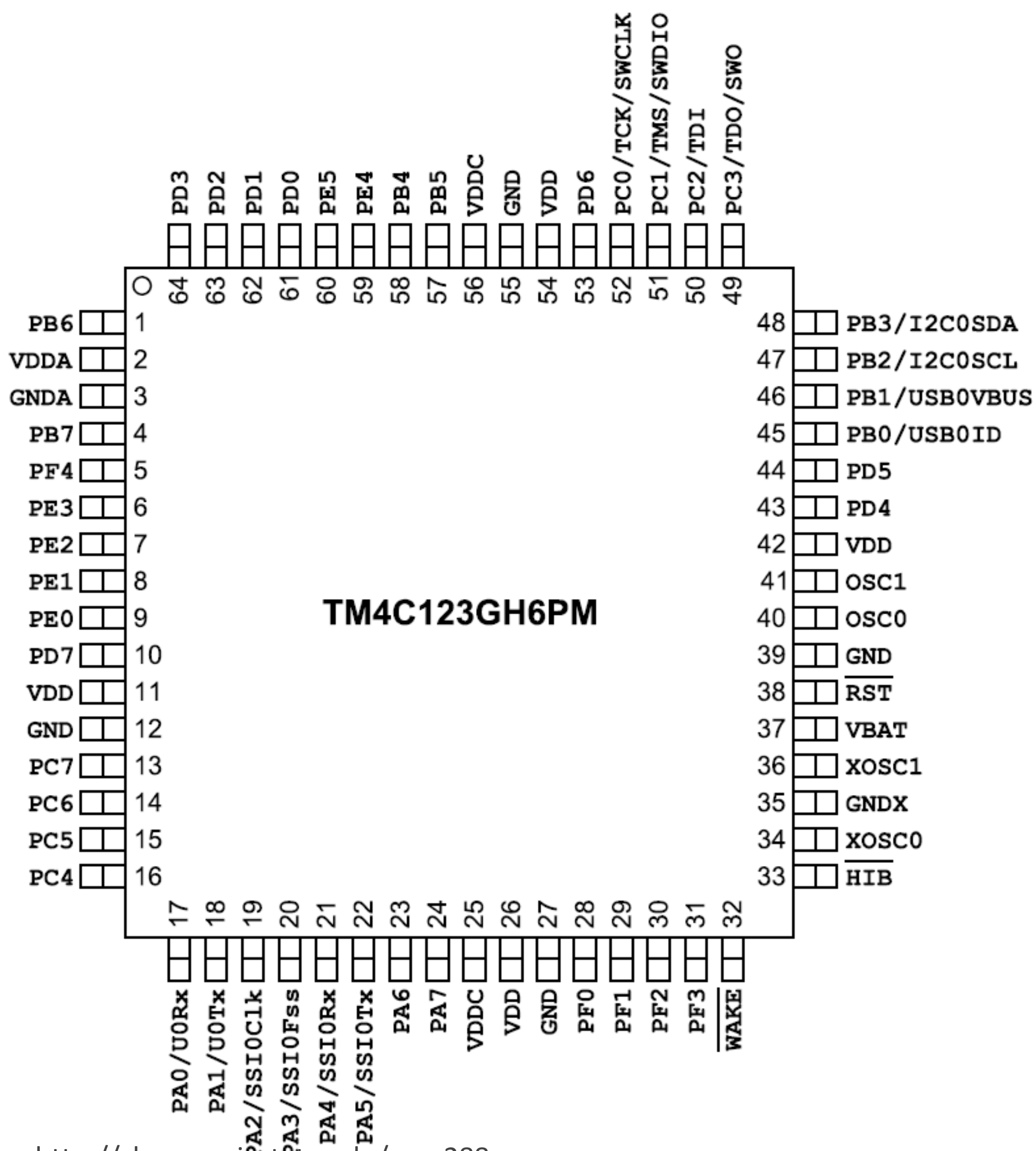
- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits



http://class.ece.iastate.edu/cpre288

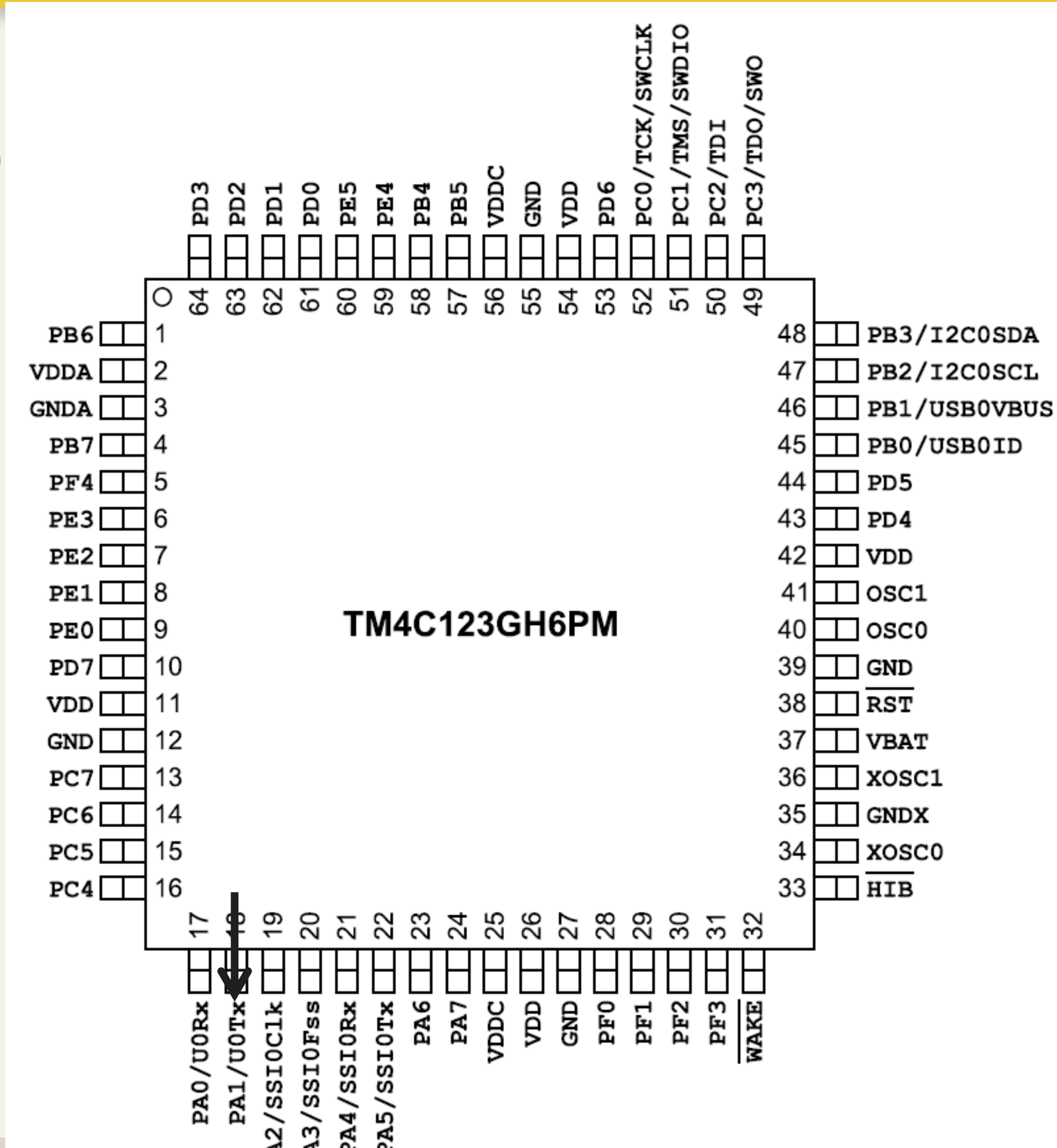
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits
- Many GPIOs have alternate functions



TM4C123 I/O Ports

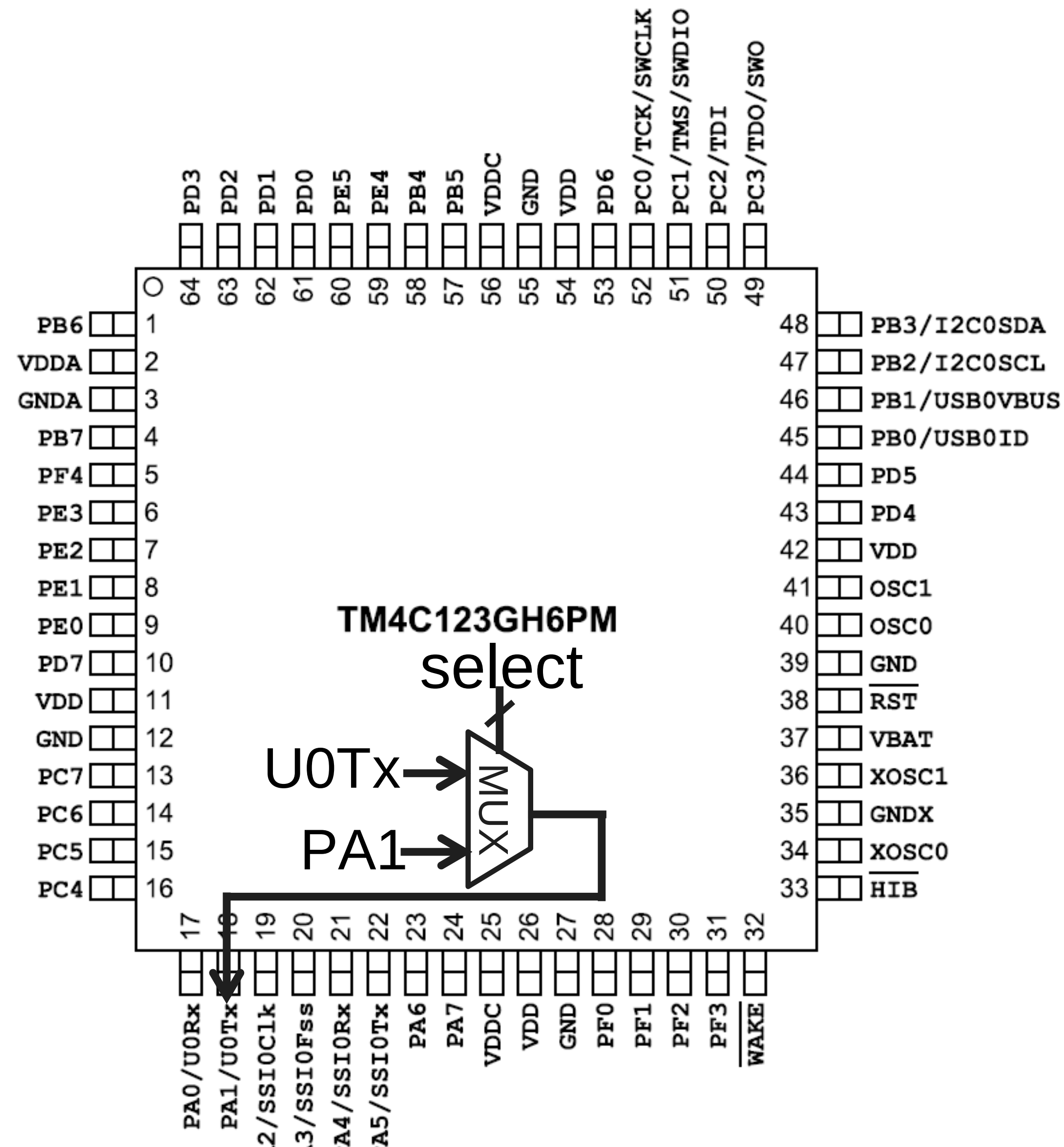
- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits
- Many GPIOs have alternate functions



http://class.ece.iastate.edu/cpre288

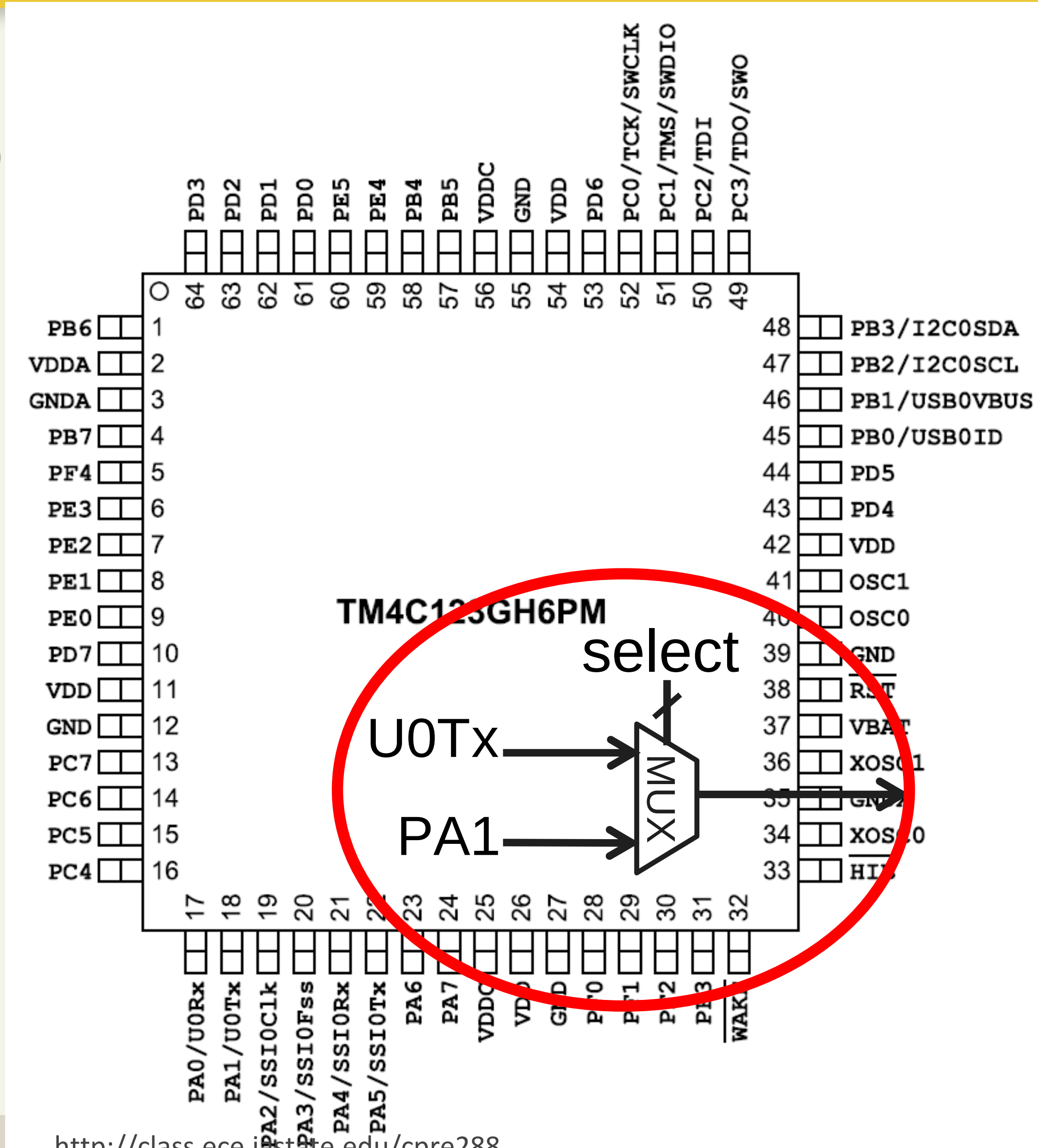
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - Each 8-bits
- Many GPIOs have alternate functions



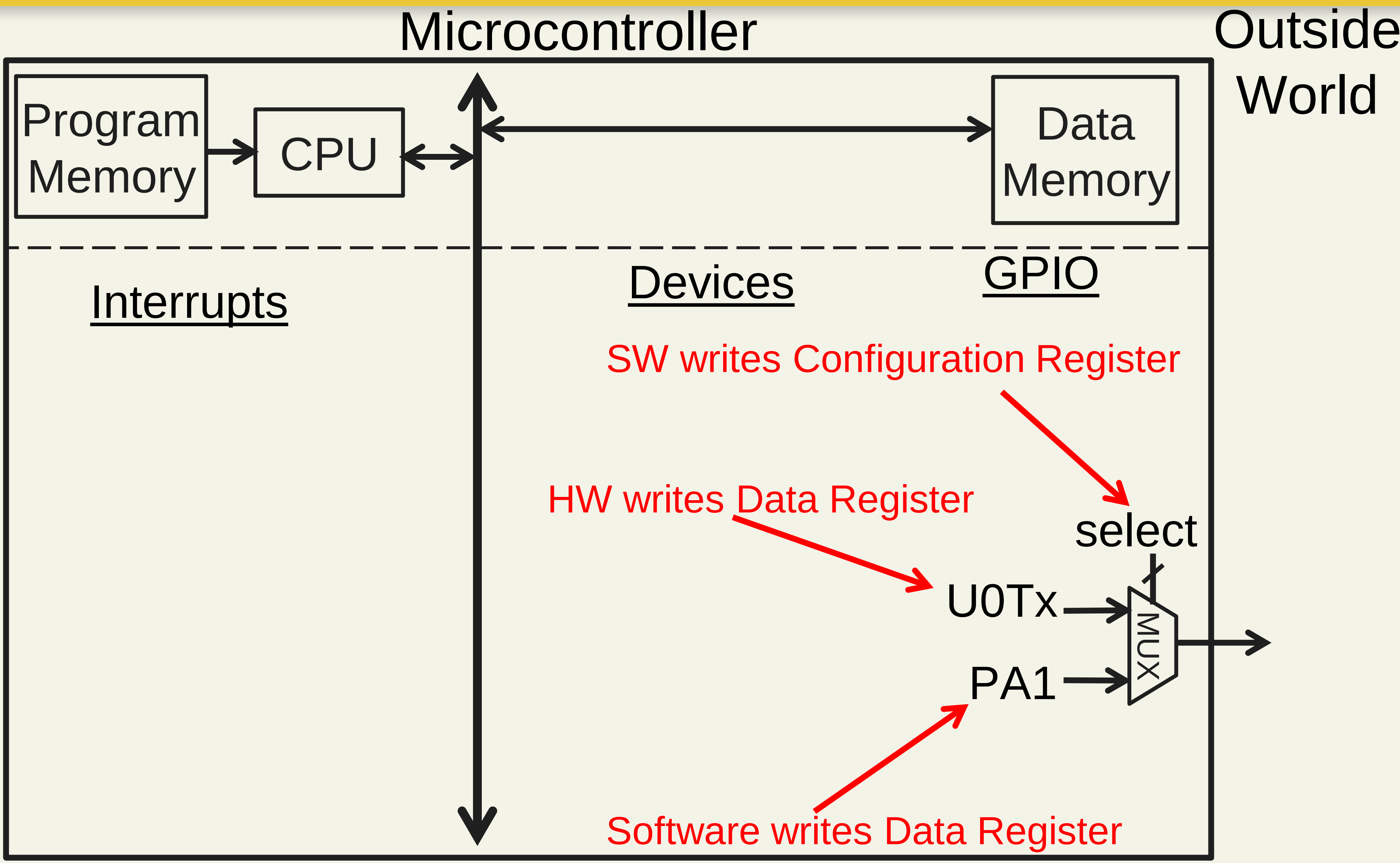
TM4C123 I/O Ports

- 64 package pins
- 6 GPIOs Ports(A – F)
  - –Each 8-bits
- Many GPIOs have alternate functions

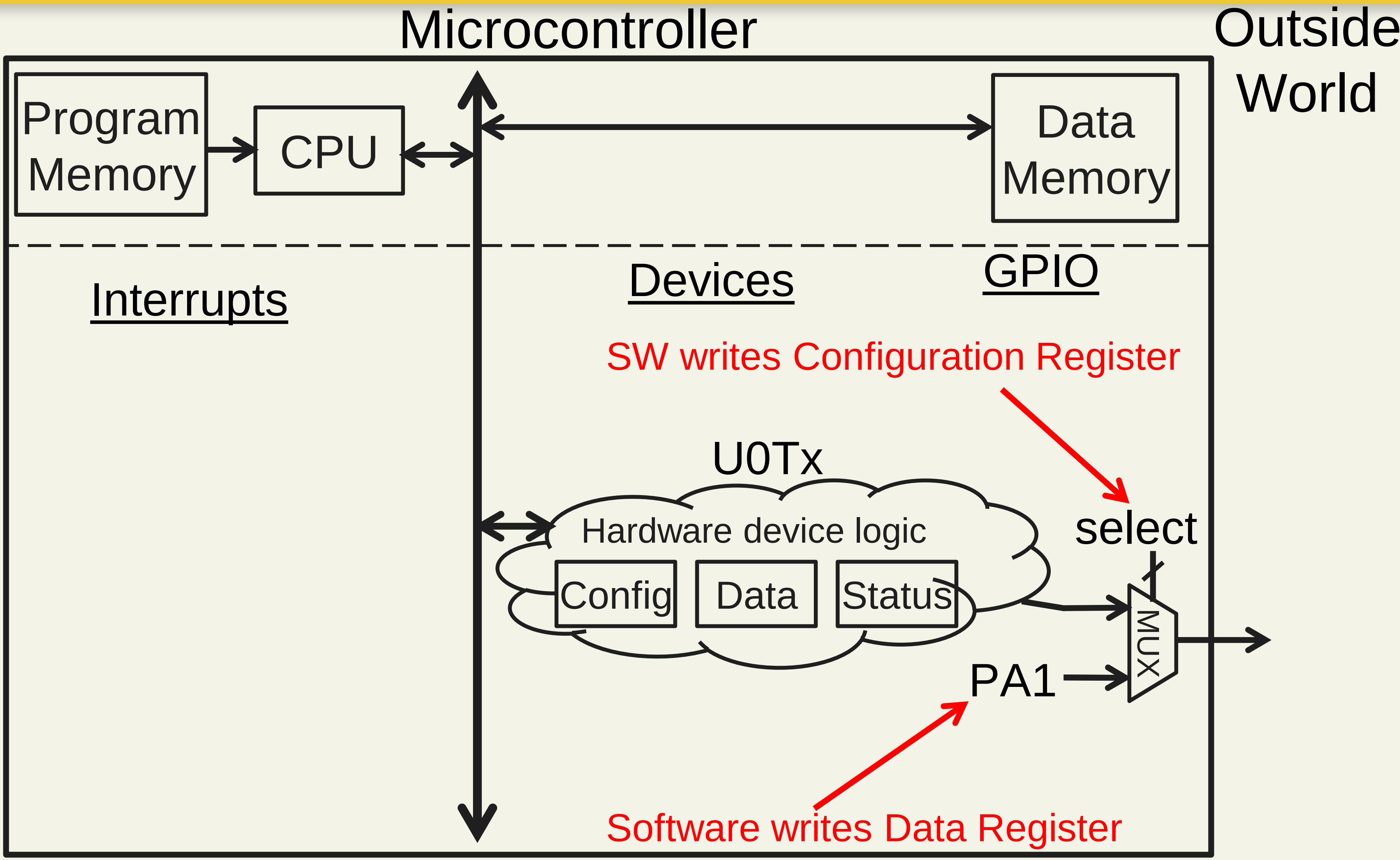


http://class.ece.iastate.edu/cpre288

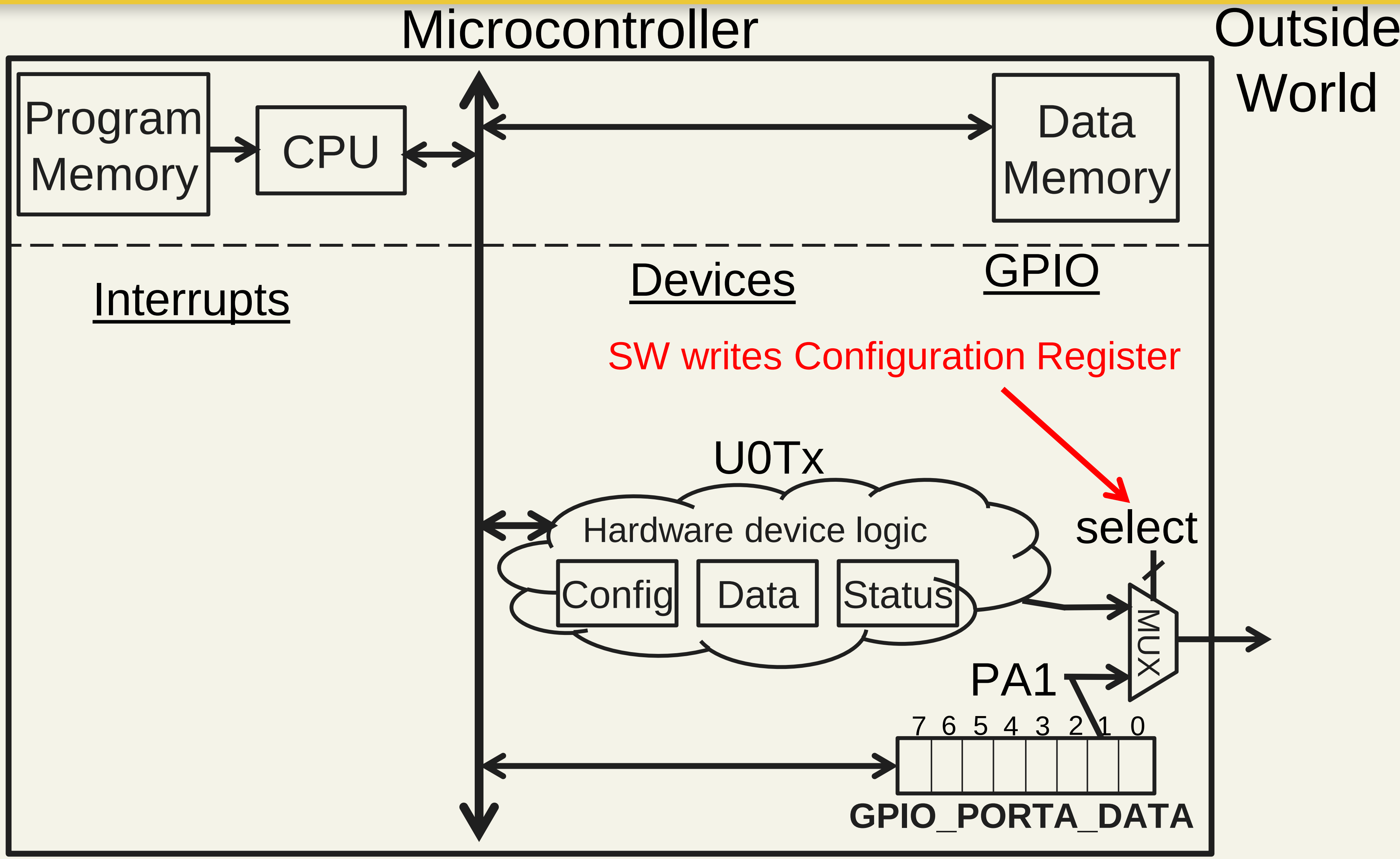
TM4C123 I/O Ports



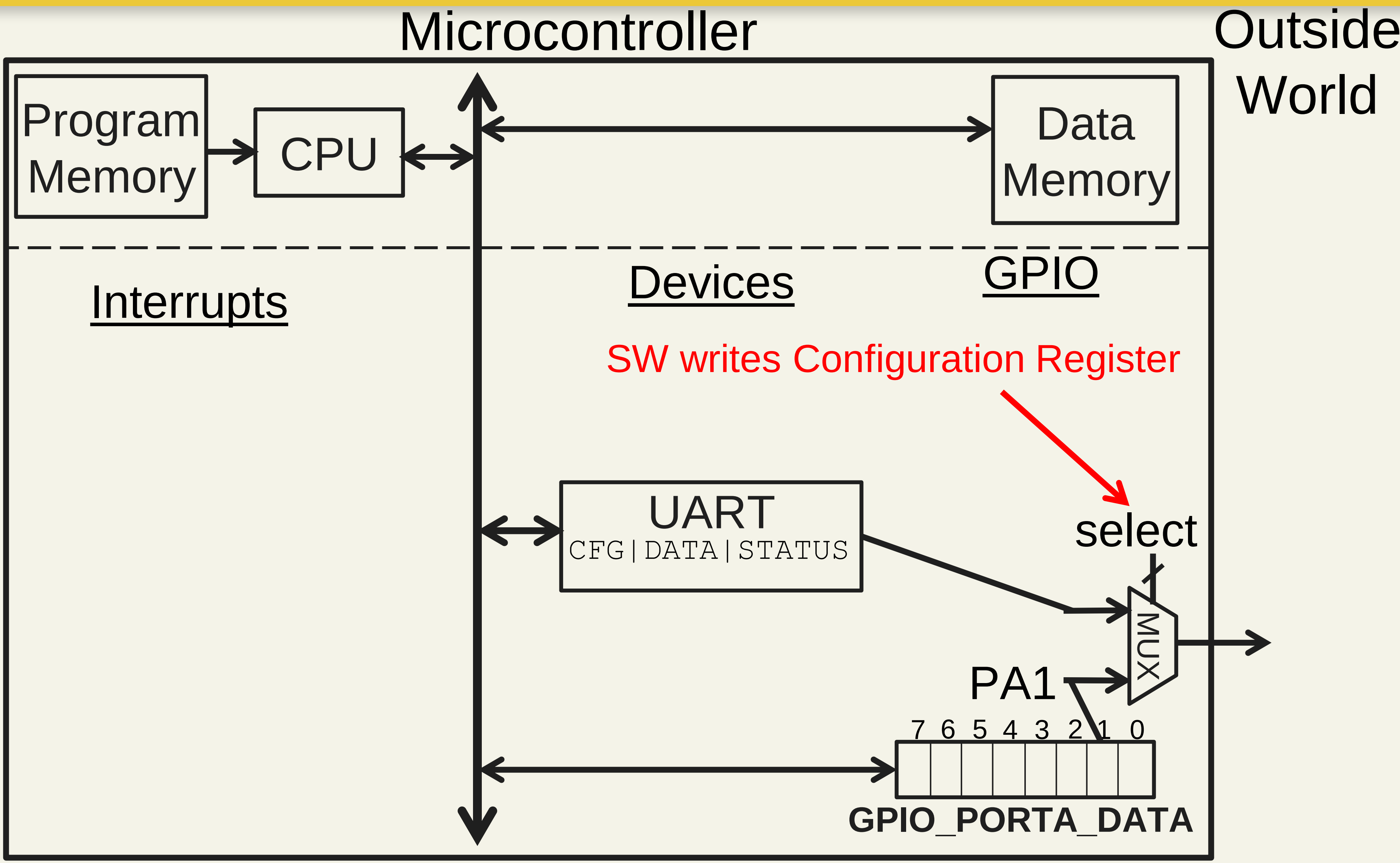
TM4C123 I/O Ports



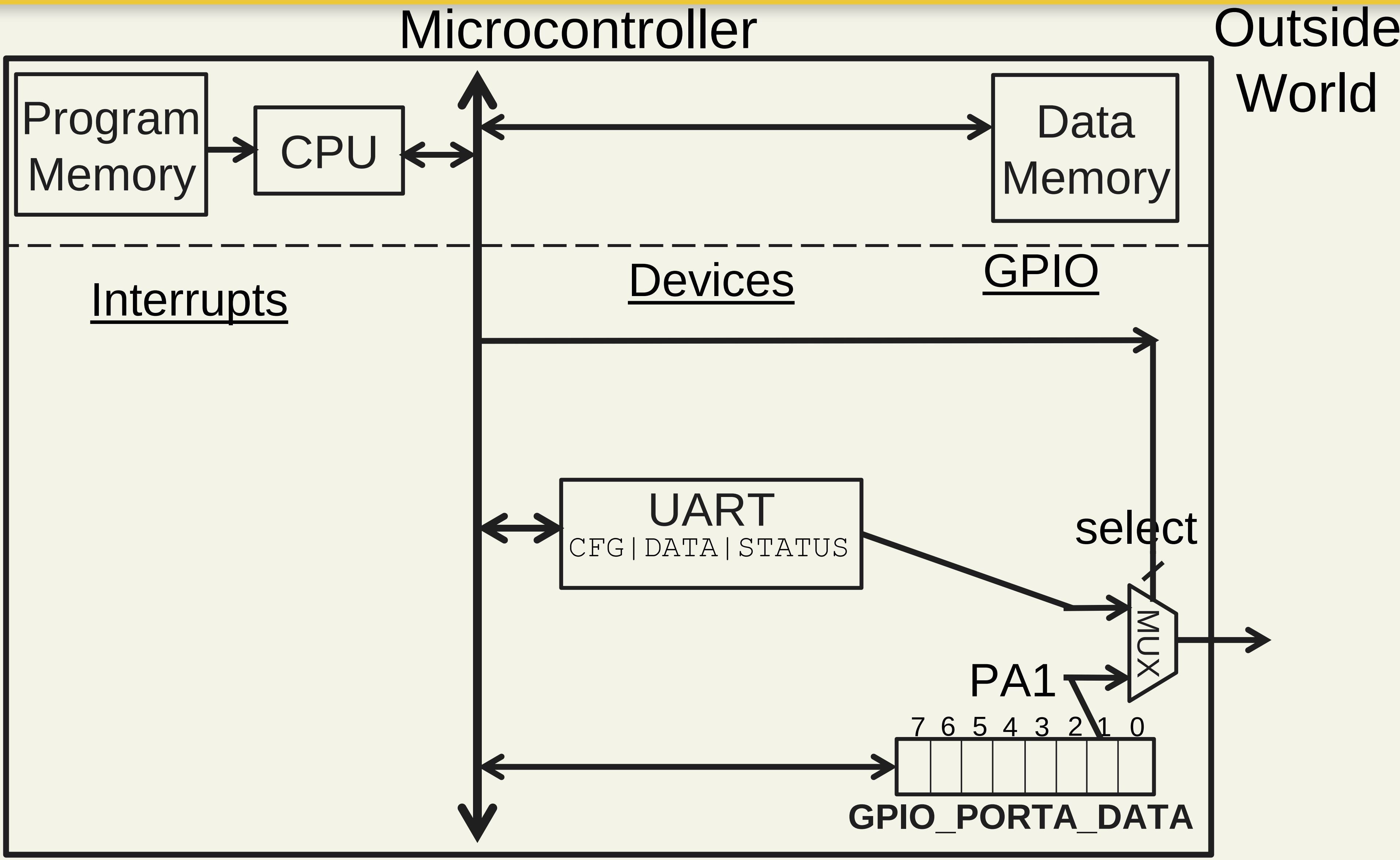
TM4C123 I/O Ports



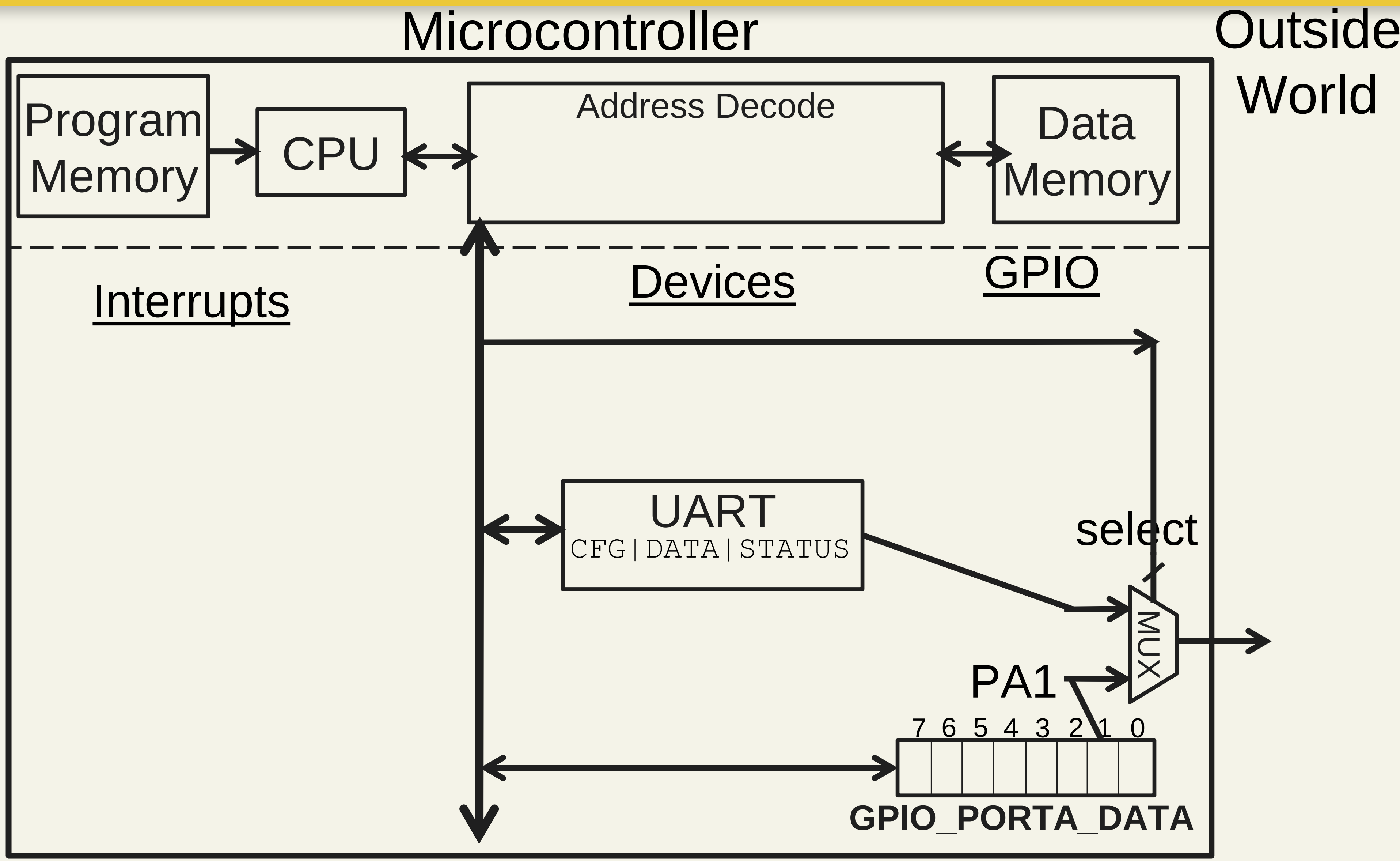
TM4C123 I/O Ports



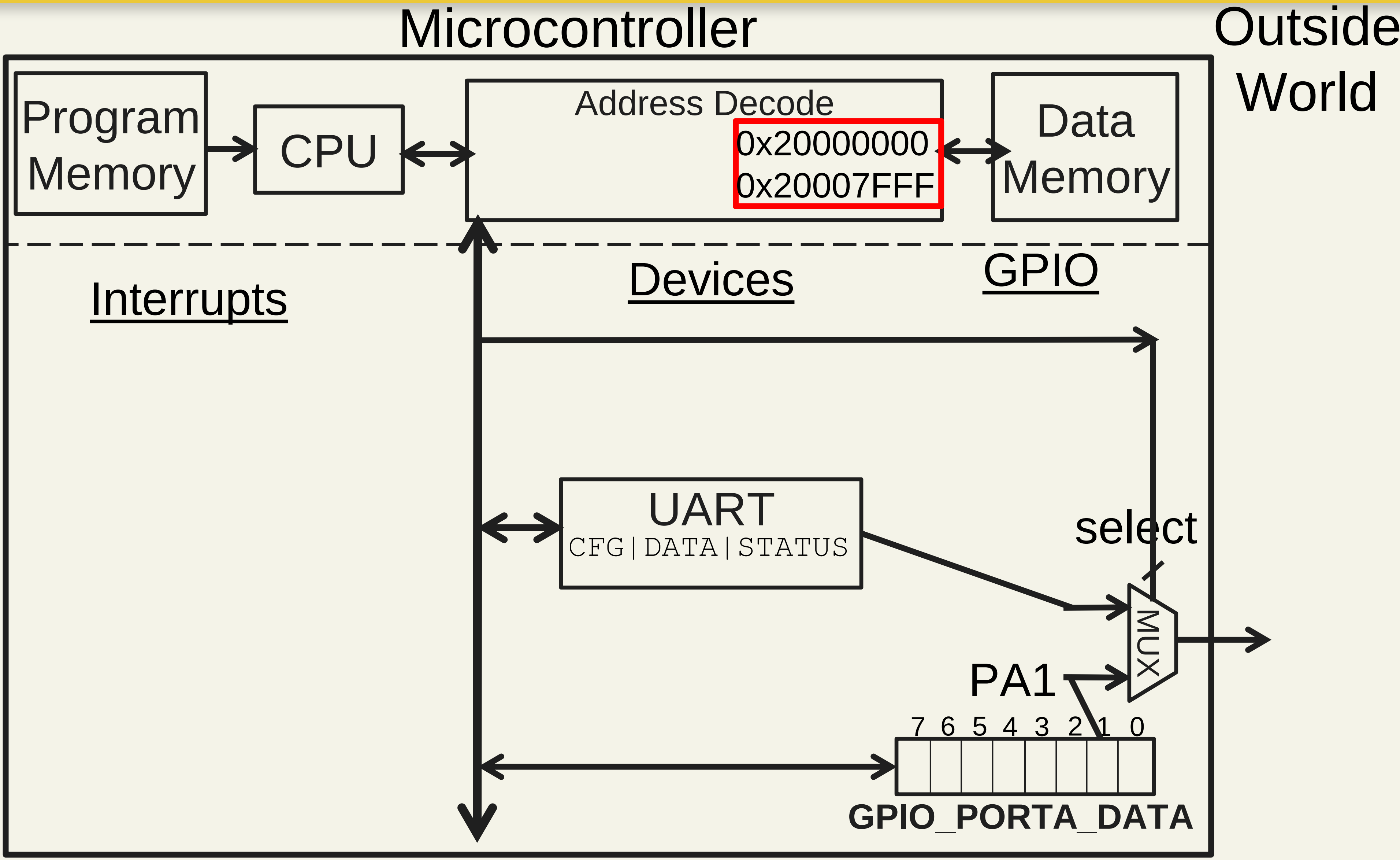
TM4C123 I/O Ports



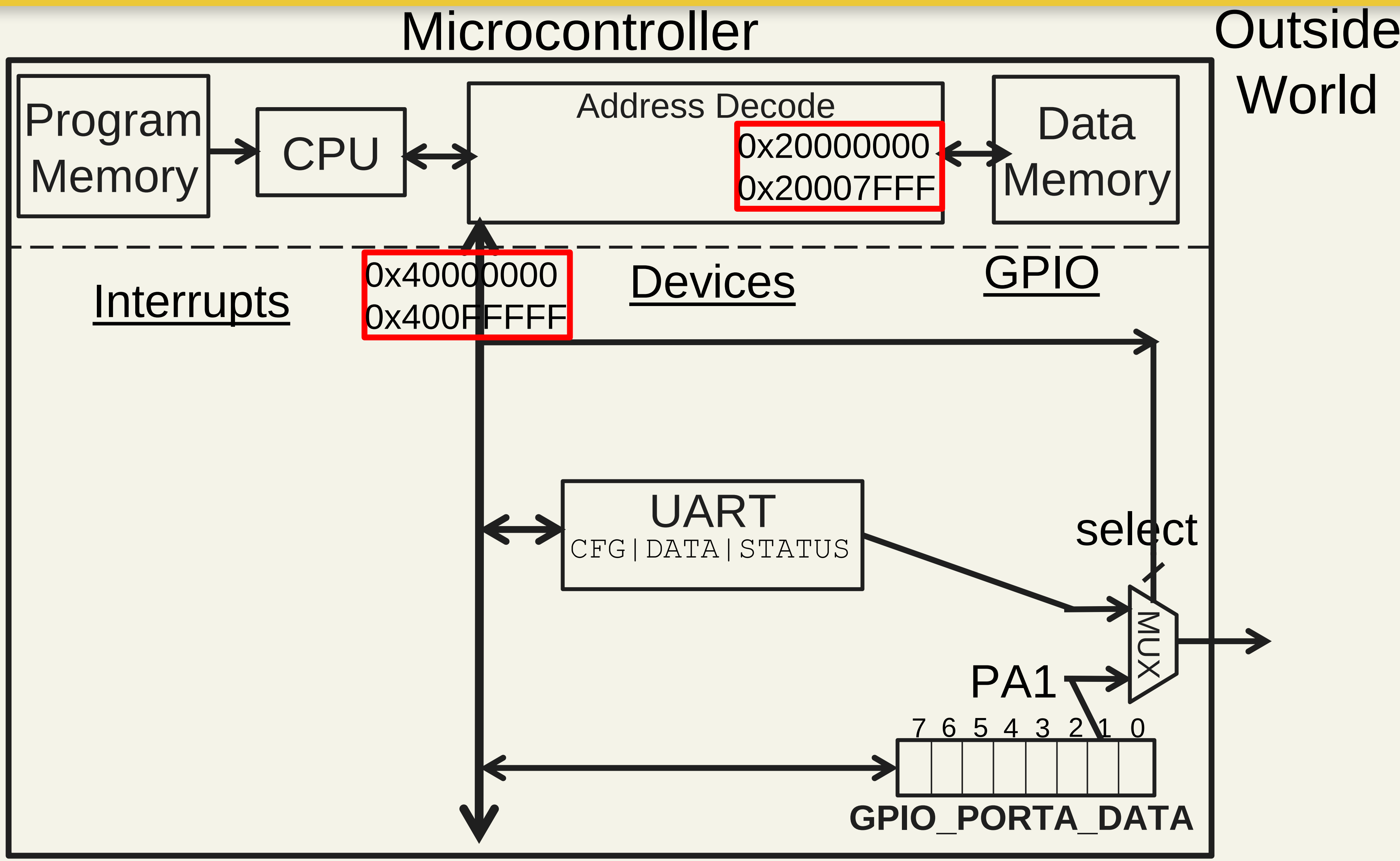
TM4C123 I/O Ports



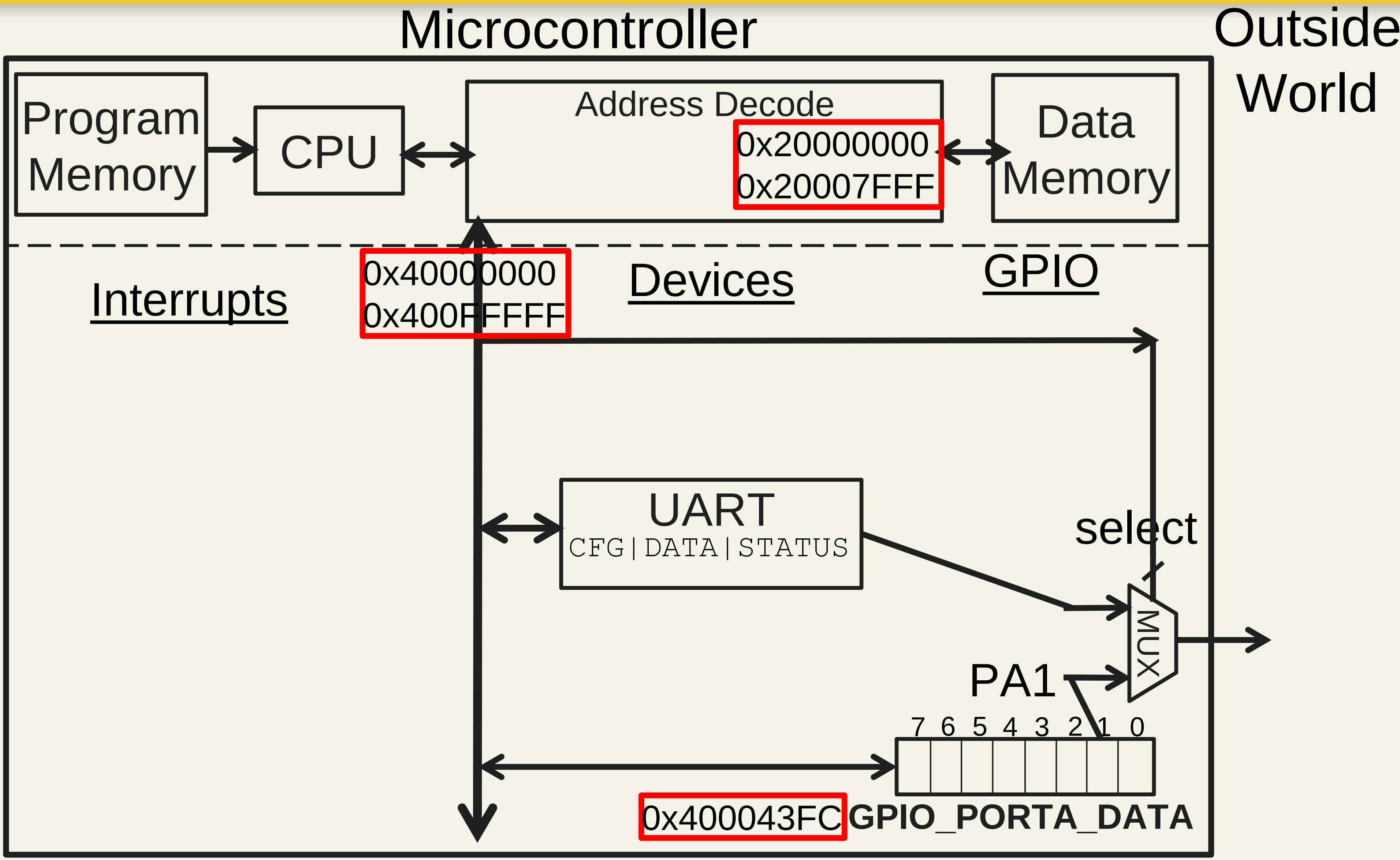
TM4C123 I/O Ports



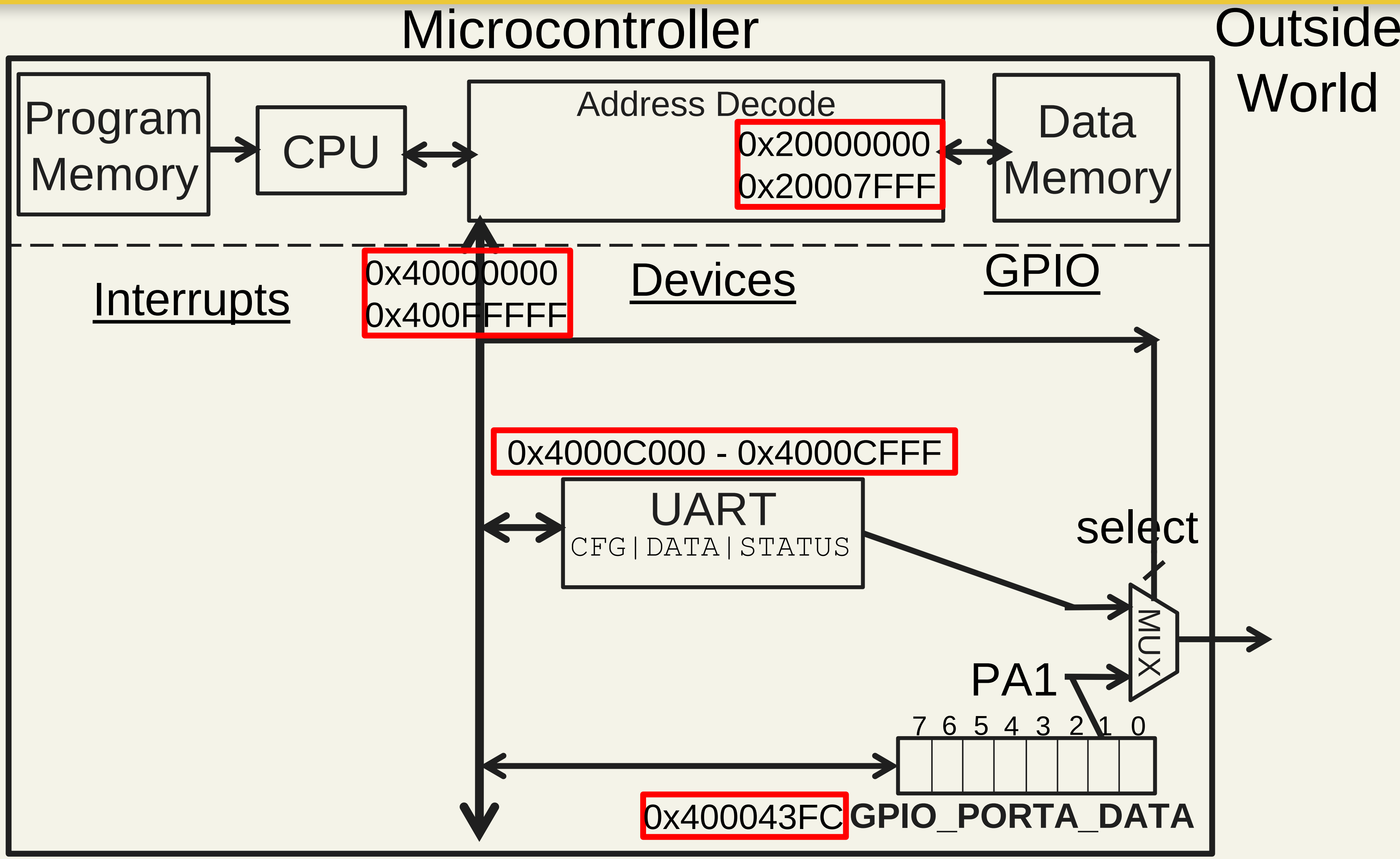
TM4C123 I/O Ports



TM4C123 I/O Ports



TM4C123 I/O Ports



CPRE 288 Microcontroller: Full Memory Map

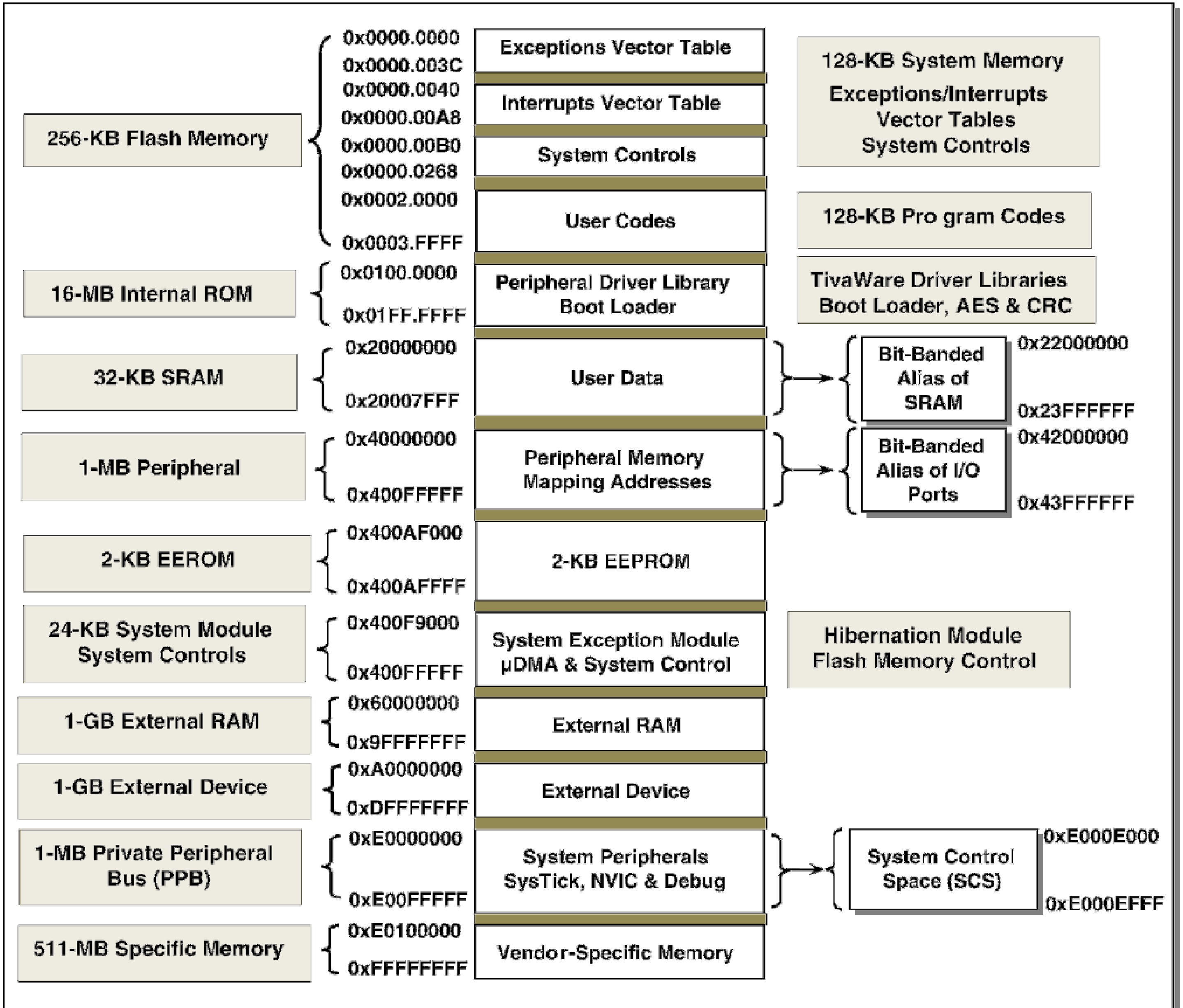
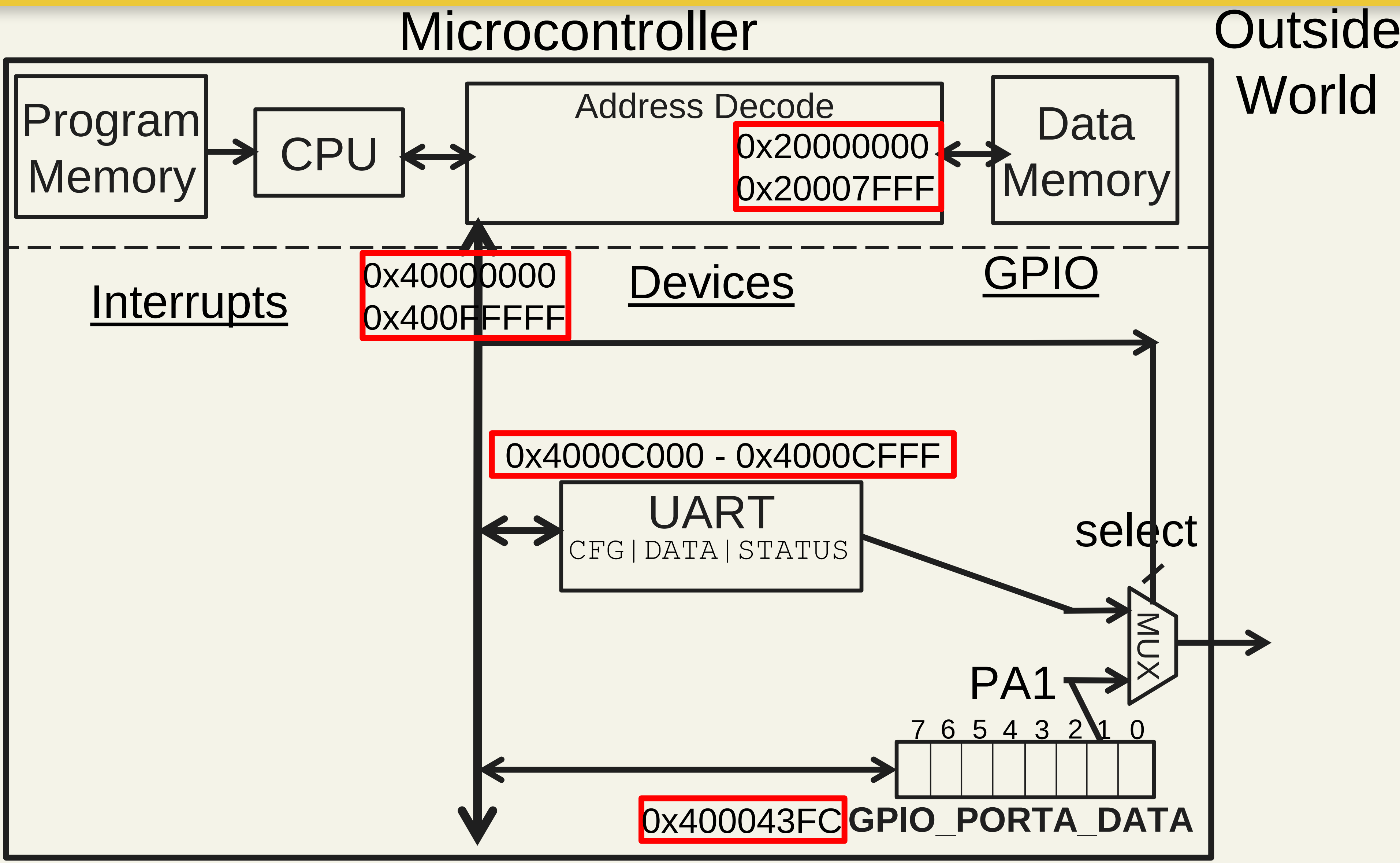


Figure 6.14. The system memory map for TM4C123GH6PM MCU.

TM4C123 I/O Ports



Memory Mapped General Purpose I/O (GPIO) Ports

TM4C123GH6PM:

- 6 general purpose I/O ports: Port A, B, C, D, E, F
- Processor communicates with them through memory mapped registers.
- A set of data and control registers associated with each port.

Memory Mapped General Purpose I/O (GPIO) Ports

- Processor communicates with arbitrary attachments using ports
- Each GPIO port has several registers, we will focus on 4.

SYSCTL_RCGCGPIO_R – Enables the port's system clock
GPIO_PORTx_DATA_R – 8-bit data register (read/write)
GPIO_PORTx_DIR_R – Data direction register
GPIO_PORTx_DEN_R – Digital enable register

Memory Mapped General Purpose I/O (GPIO) Ports

SYSCTL_RCGCGPIO_R (GPIO Run Mode Clock Gating Control)

- A GPIO port's clock must be enabled to use its registers.

(Your program will crash if you attempt to update an GPIO port's registers without enabling its clock)

- bits 5-0 in SYSCTL_RCGCGPIO_R represent ports F-A.

E.g.:

```
//Enable Port F & B clocks
SYSCTL_RCGCGPIO_R |= 0b100010;
```

Memory Mapped General Purpose I/O (GPIO) Ports

GPIO_PORTx_DEN_R (GPIO Digital Enable Register)

- Writing `‘0’` to a bit disables the digital functionality for the corresponding pin of the Port.
- Writing `‘1’` to a bit enables the digital functionality for the corresponding pin of the Port.

Example:

```
// Enable Digital functionality for pin
// 7 and 0 of Port A, and preserve the others
GPIO_PORTA_DEN_R |= 0b10000001;
```

Memory Mapped General Purpose I/O (GPIO) Ports

GPIO_PORTx_DIR_R (GPIO Direction Register)

- Writing `'0'` to a bit programs the corresponding pin of the Port as input
- Writing `'1'` to a bit programs the corresponding pin of the Port as output

Example:

```
//All bits of port A prog for input except bit0
GPIO_PORTA_DIR_R = 0b00000001;
```

Memory Mapped General Purpose I/O (GPIO) Ports

GPIO_PORTX_DATA_R Register:

For output configured port: If a bit in in the Port's DATA register is written when the corresponding pin is configured as an output, then the port's pin will be driven with this value.

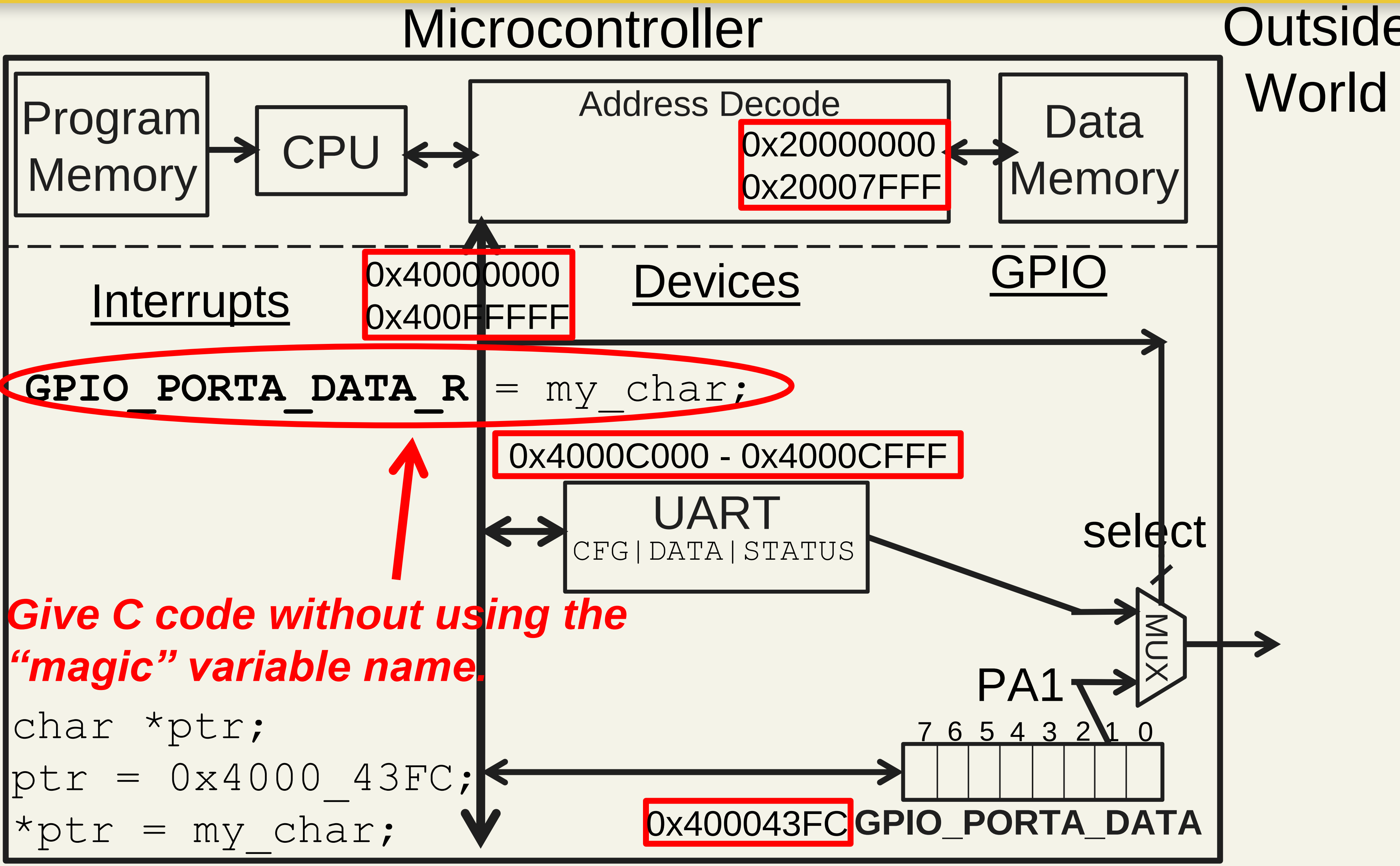
Write to a port using its DATA register.

E.g.:

```
char my_char = 0b1111_0000;
SYSCTL_RCGCGPIO_R |= 0b000001; //enable PORTA clock
GPIO_PORTA_DIR_R = 0xFF; //set port A dir to output
GPIO_PORTA_DEN_R = 0xFF; //Enable pins 0-7
GPIO_PORTA_DATA_R = my_char; //set port A to my_char
```

Give C code without using the "magic" variable name.

TM4C123 I/O Ports



Give C code without using the "magic" variable name.

```
char *ptr;
ptr = 0x4000_43FC;
*ptr = my_char;
```

Memory Mapped General Purpose I/O (GPIO) Ports

GPIO_PORTX_DATA_R Register:

For output configured port: If a bit in in the Port's DATA register is written when the corresponding pin is configured as an output, then the port's pin will be driven with this value.

Write to a port using its DATA register.

E.g.:

```
char my_char = 0b1111_0000;
SYSCTL_RCGCGPIO_R |= 0b000001; //enable PORTA clock
GPIO_PORTA_DIR_R = 0xFF; //set port A dir to output
GPIO_PORTA_DEN_R = 0xFF; //Enable pins 0-7
GPIO_PORTA_DATA_R = my_char;//set port A to my_char
```

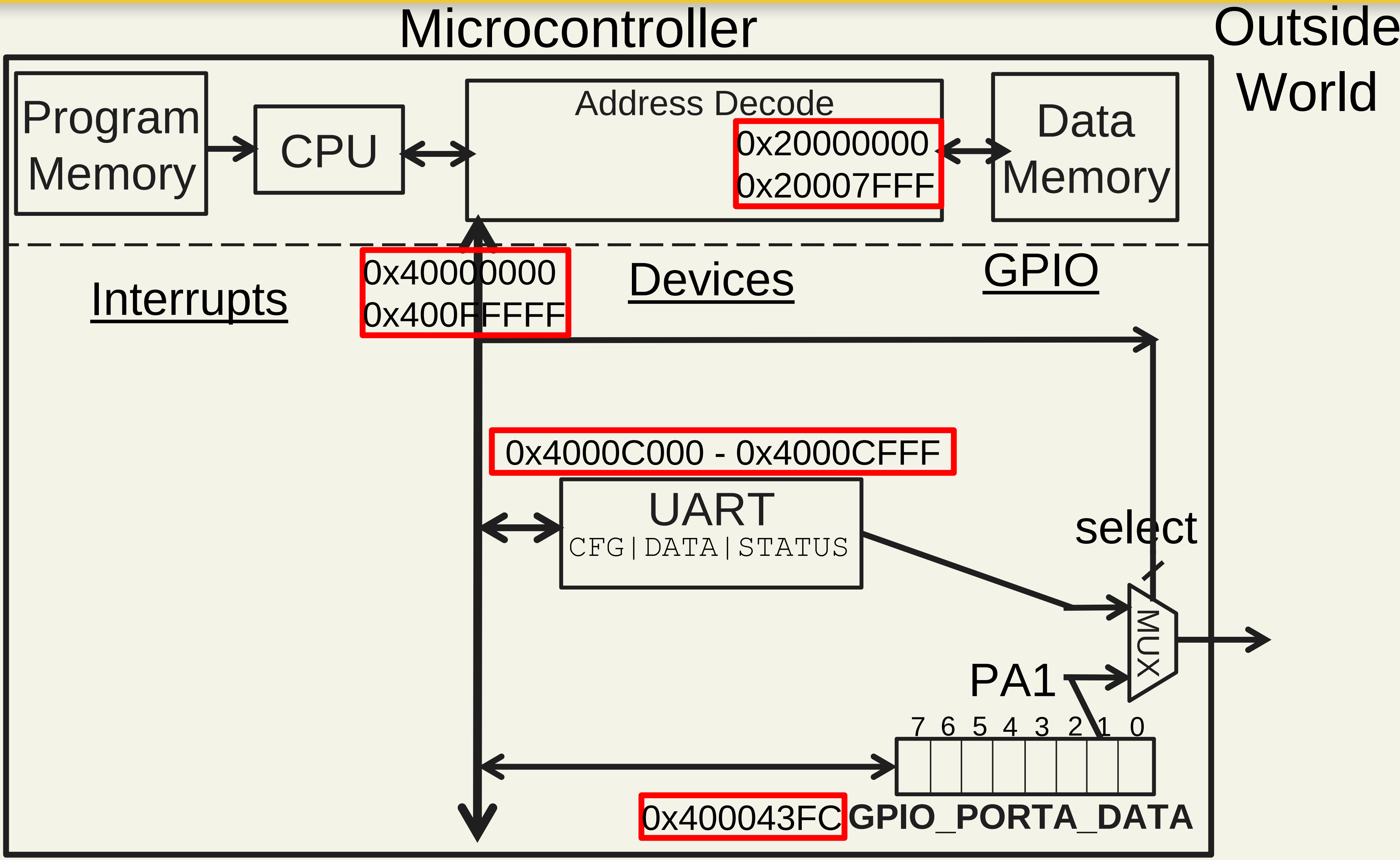
Directly using the definition of **GPIO_PORTA_DATA_R**

```
*((char *)0x400043FC) = my_char;
```

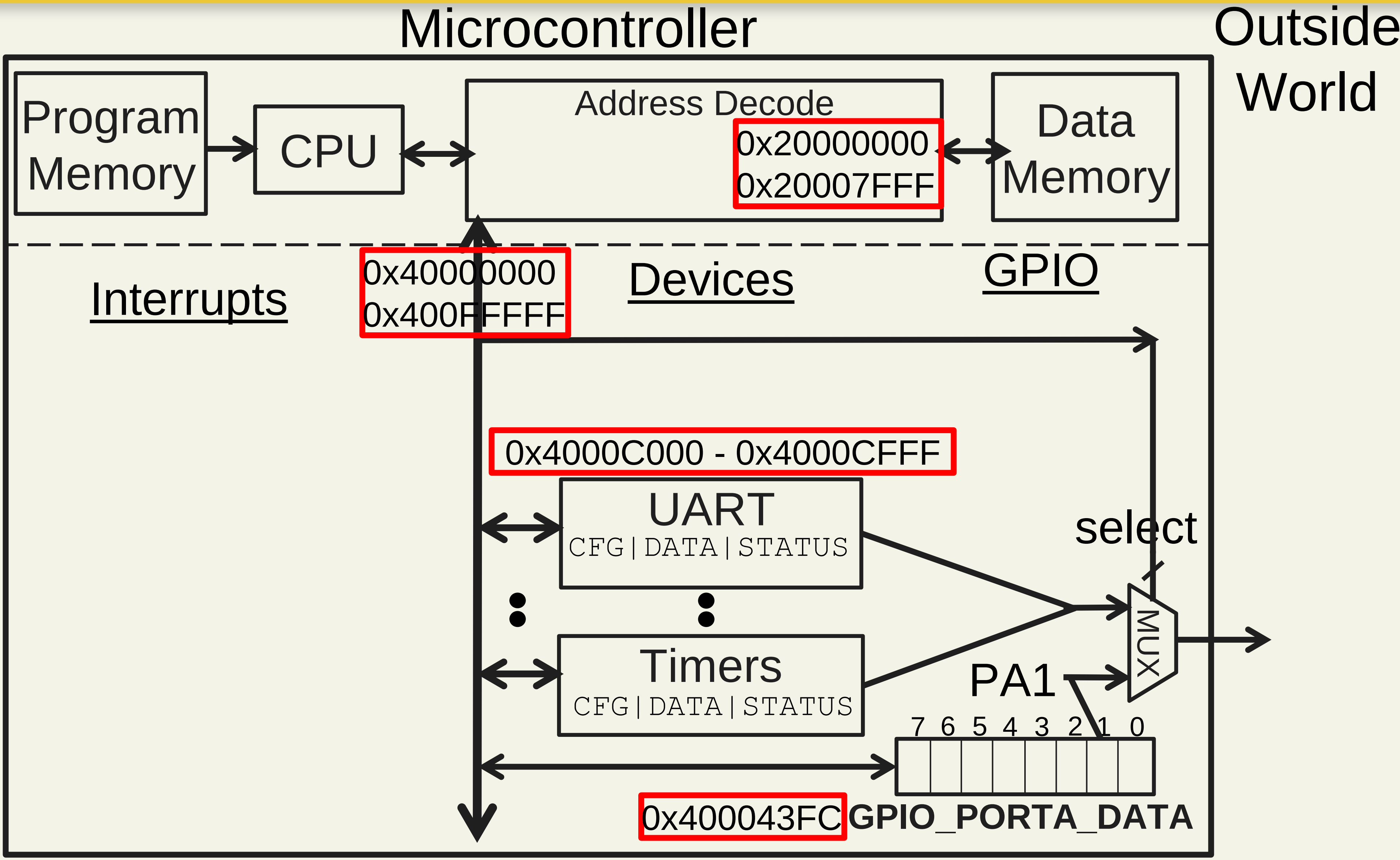
Memory Mapped General Purpose I/O (GPIO) Ports

```
// tm4c123gh6pm.h
//*****************************************************************************
// GPIO registers (PORTA)
//*****************************************************************************
#define GPIO_PORTA_DATA_BITS_R  ((volatile unsigned long *)0x40004000)
#define GPIO_PORTA_DATA_R       (*((volatile unsigned long *)0x400043FC))
#define GPIO_PORTA_DIR_R        (*((volatile unsigned long *)0x40004400))
#define GPIO_PORTA_IS_R         (*((volatile unsigned long *)0x40004404))
#define GPIO_PORTA_IBE_R        (*((volatile unsigned long *)0x40004408))
#define GPIO_PORTA_IEV_R        (*((volatile unsigned long *)0x4000440C))
#define GPIO_PORTA_IM_R         (*((volatile unsigned long *)0x40004410))
#define GPIO_PORTA_RIS_R        (*((volatile unsigned long *)0x40004414))
#define GPIO_PORTA_MIS_R        (*((volatile unsigned long *)0x40004418))
#define GPIO_PORTA_ICR_R        (*((volatile unsigned long *)0x4000441C))
#define GPIO_PORTA_AFSEL_R      (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTA_DR2R_R       (*((volatile unsigned long *)0x40004500))
#define GPIO_PORTA_DR4R_R       (*((volatile unsigned long *)0x40004504))
#define GPIO_PORTA_DR8R_R       (*((volatile unsigned long *)0x40004508))
#define GPIO_PORTA_ODR_R        (*((volatile unsigned long *)0x4000450C))
#define GPIO_PORTA_PUR_R        (*((volatile unsigned long *)0x40004510))
#define GPIO_PORTA_PDR_R        (*((volatile unsigned long *)0x40004514))
#define GPIO_PORTA_SLR_R        (*((volatile unsigned long *)0x40004518))
#define GPIO_PORTA_DEN_R        (*((volatile unsigned long *)0x4000451C))
#define GPIO_PORTA_LOCK_R       (*((volatile unsigned long *)0x40004520))
#define GPIO_PORTA_CR_R         (*((volatile unsigned long *)0x40004524))
#define GPIO_PORTA_AMSEL_R      (*((volatile unsigned long *)0x40004528))
#define GPIO_PORTA_PCTL_R       (*((volatile unsigned long *)0x4000452C))
#define GPIO_PORTA_ADCCTL_R     (*((volatile unsigned long *)0x40004530))
#define GPIO_PORTA_DMACTL_R     (*((volatile unsigned long *)0x40004534))
```

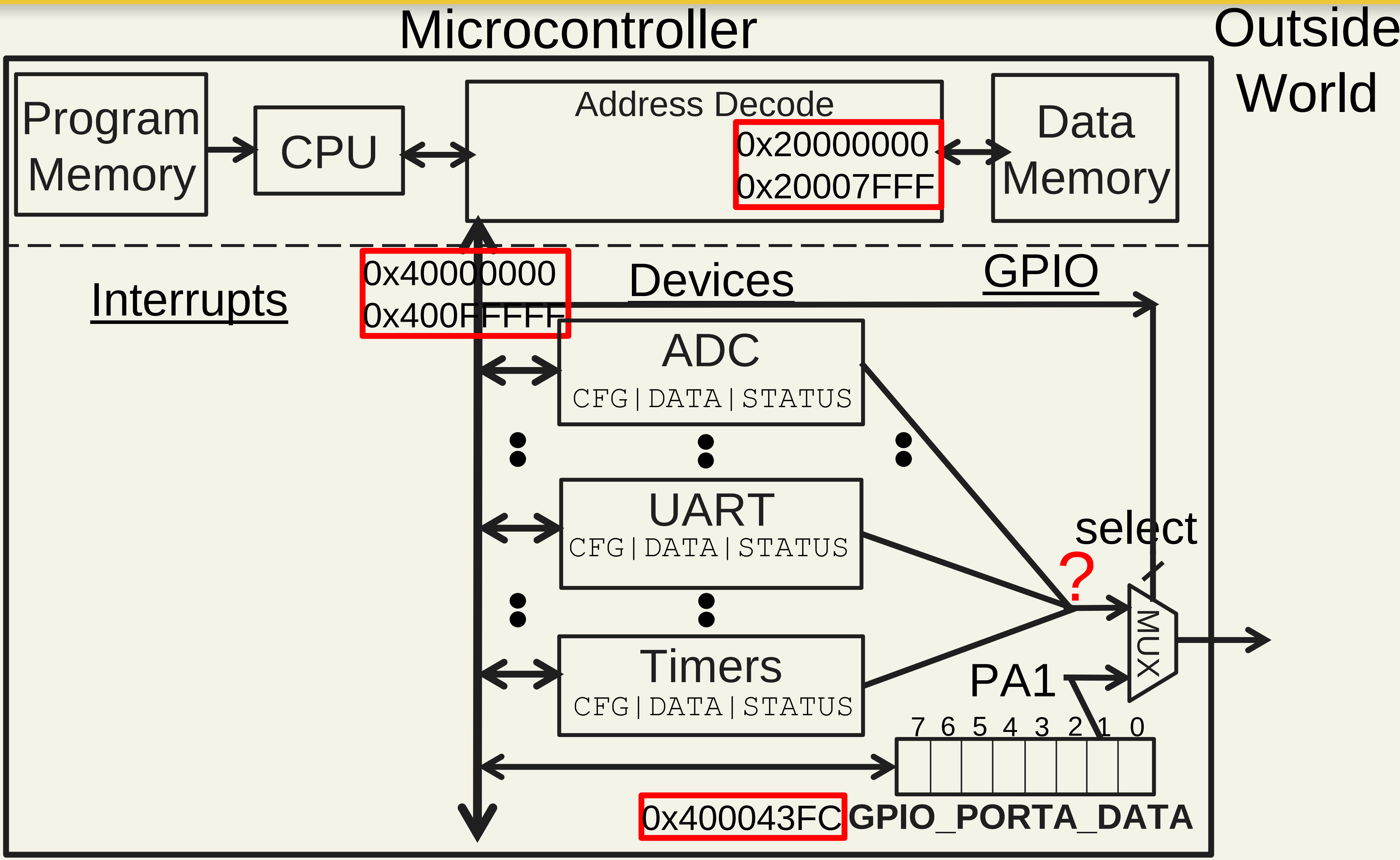
TM4C123 I/O Ports



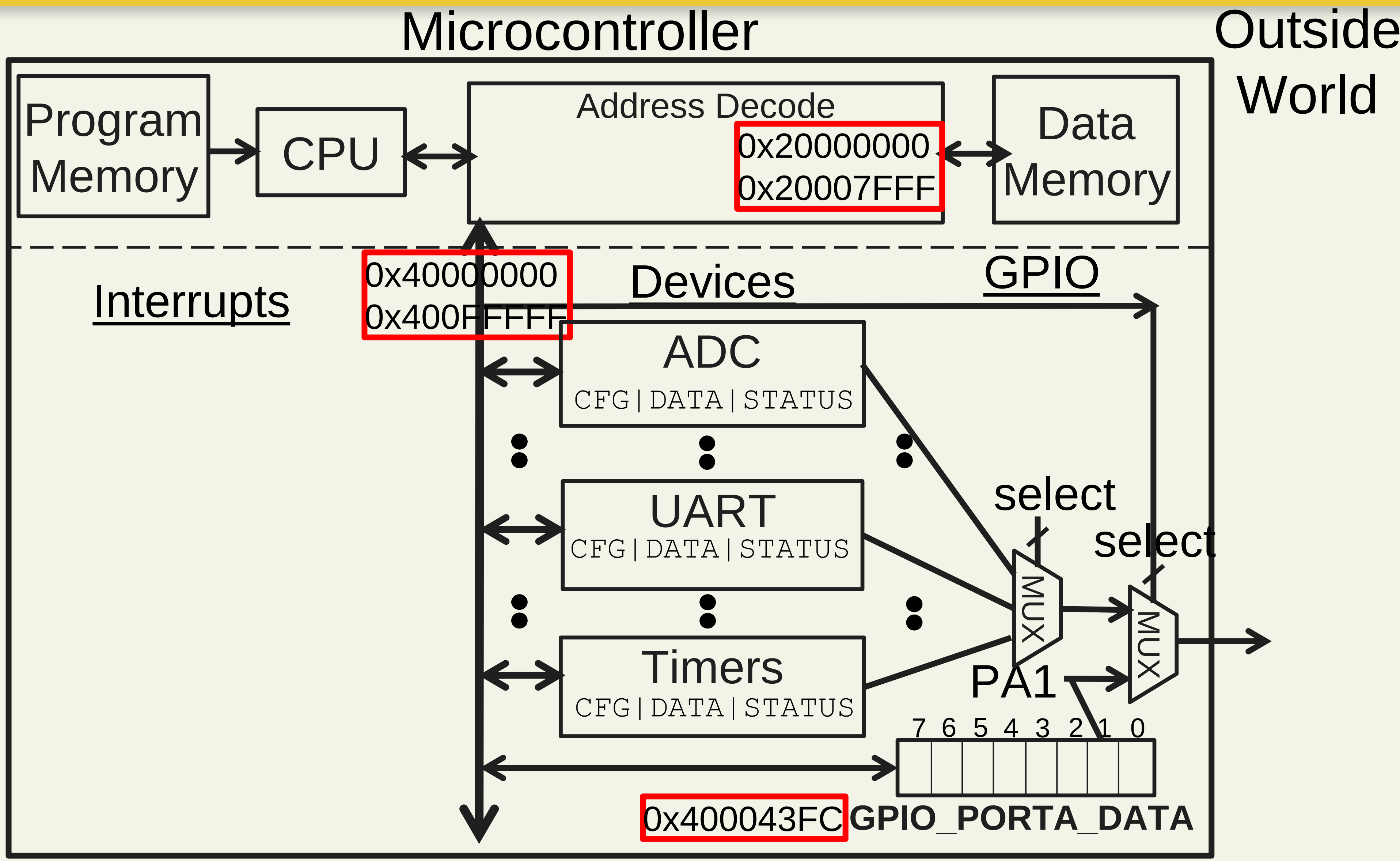
TM4C123 I/O Ports



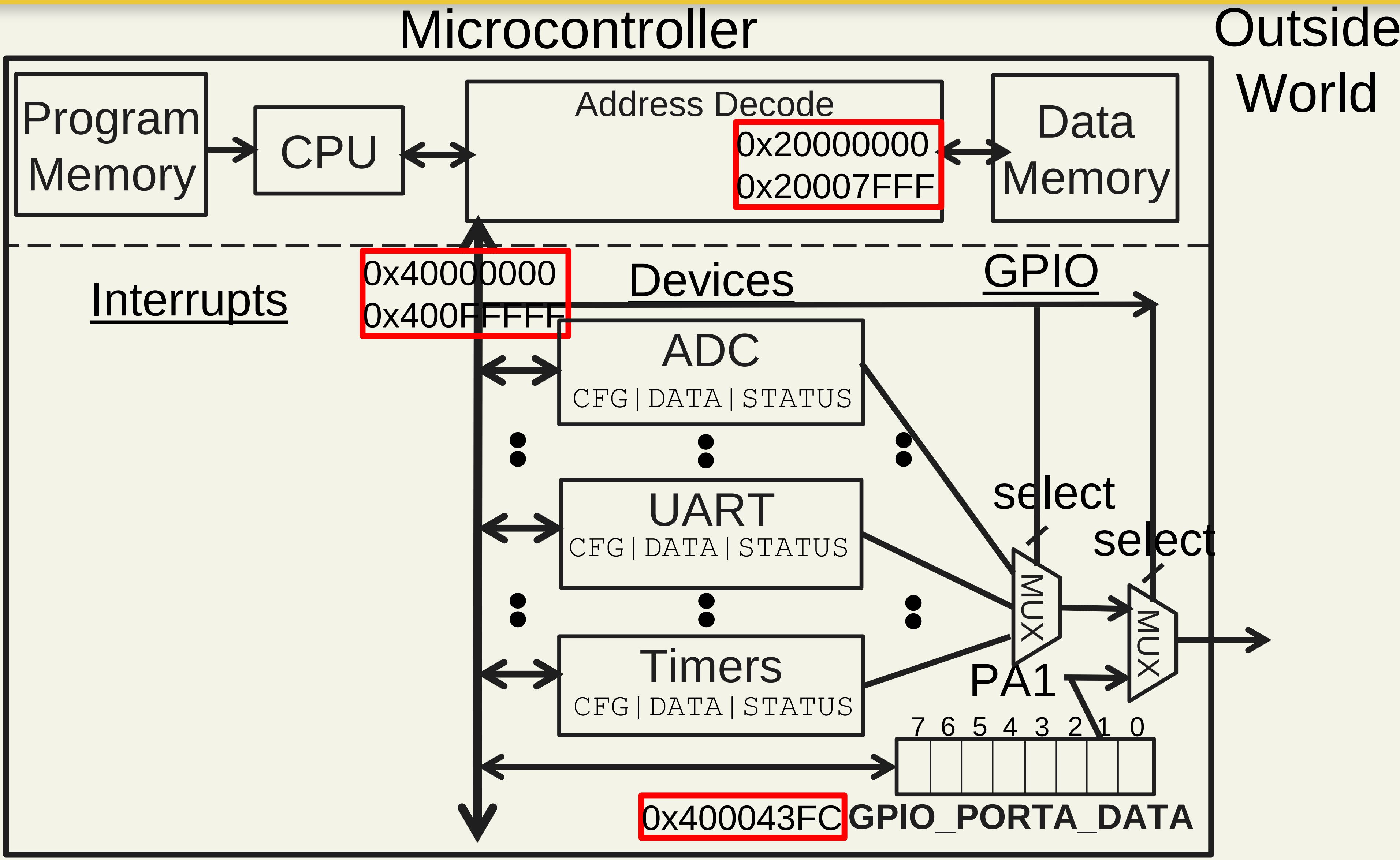
TM4C123 I/O Ports



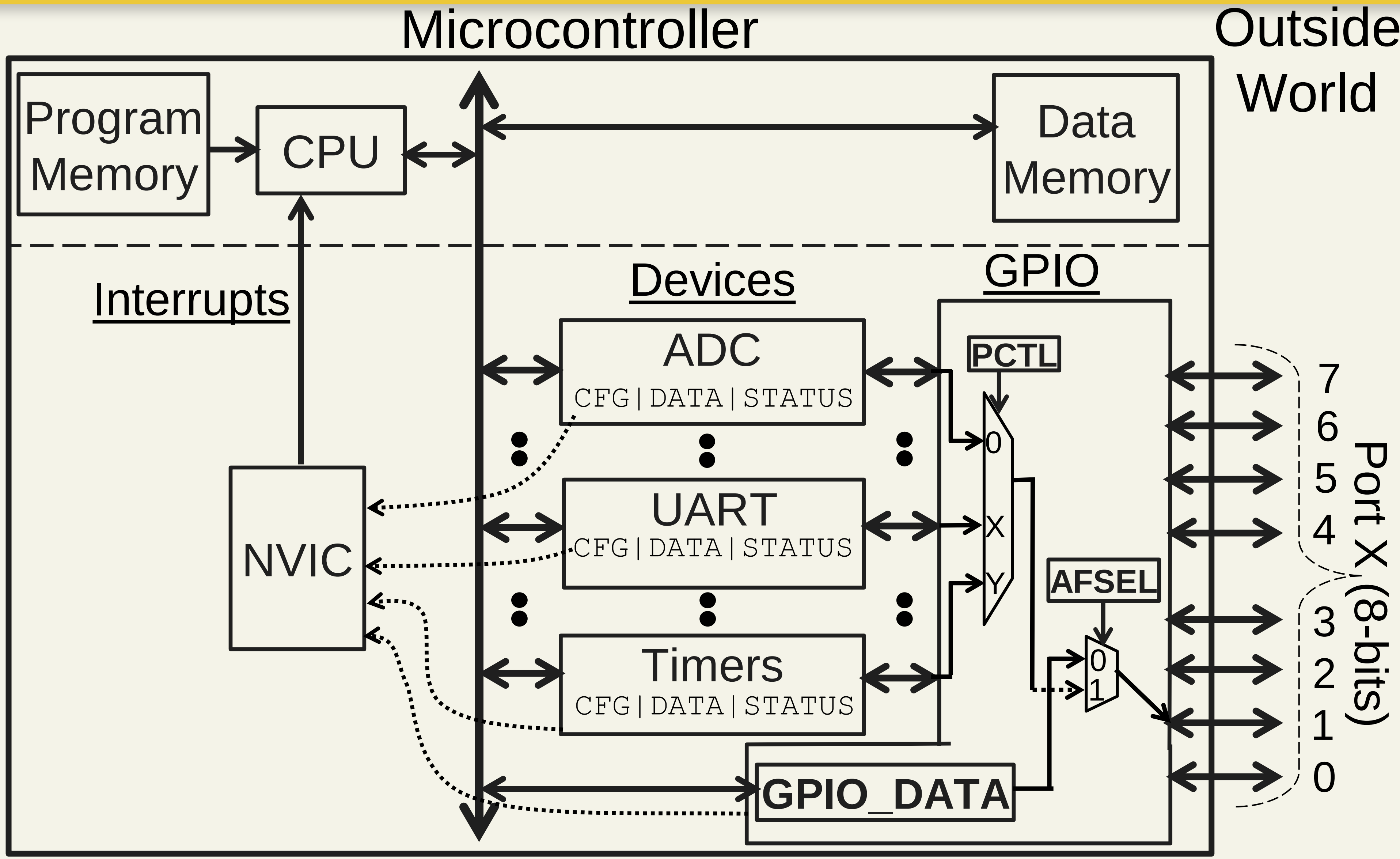
TM4C123 I/O Ports



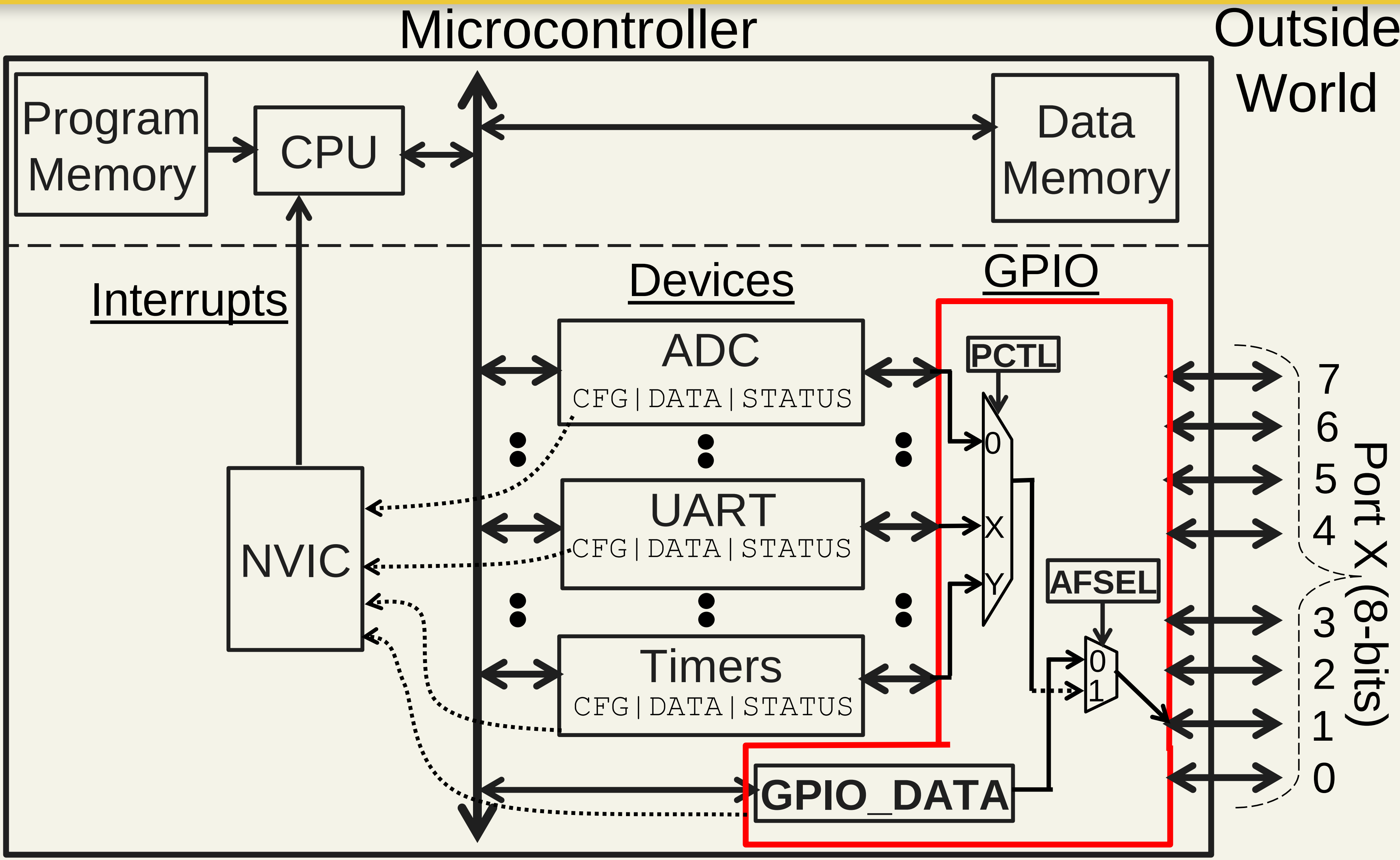
TM4C123 I/O Ports



Microcontroller / System-on-Chip (SoC)



Microcontroller / System-on-Chip (SoC)



GPIO Port: Alternative Function Architecture

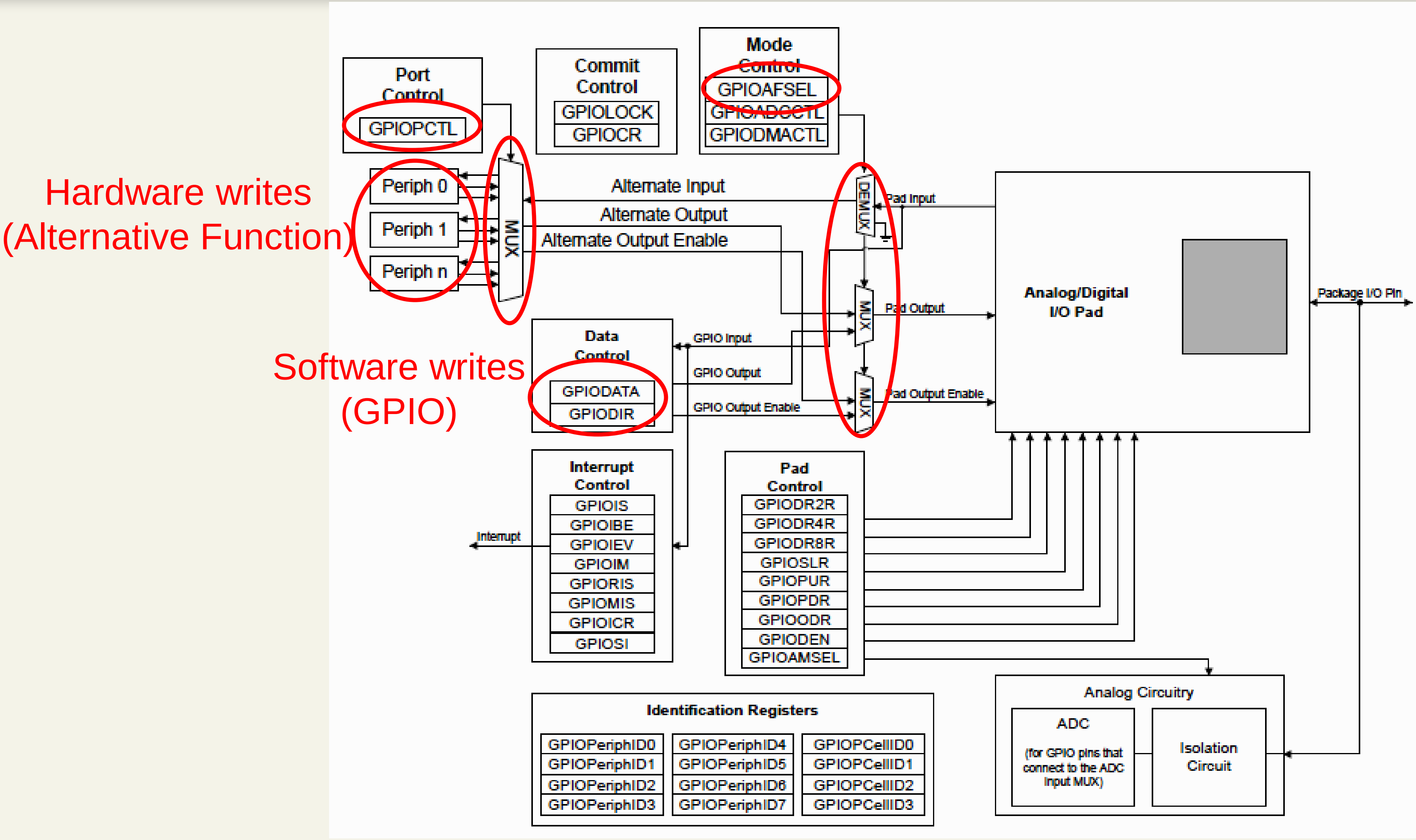
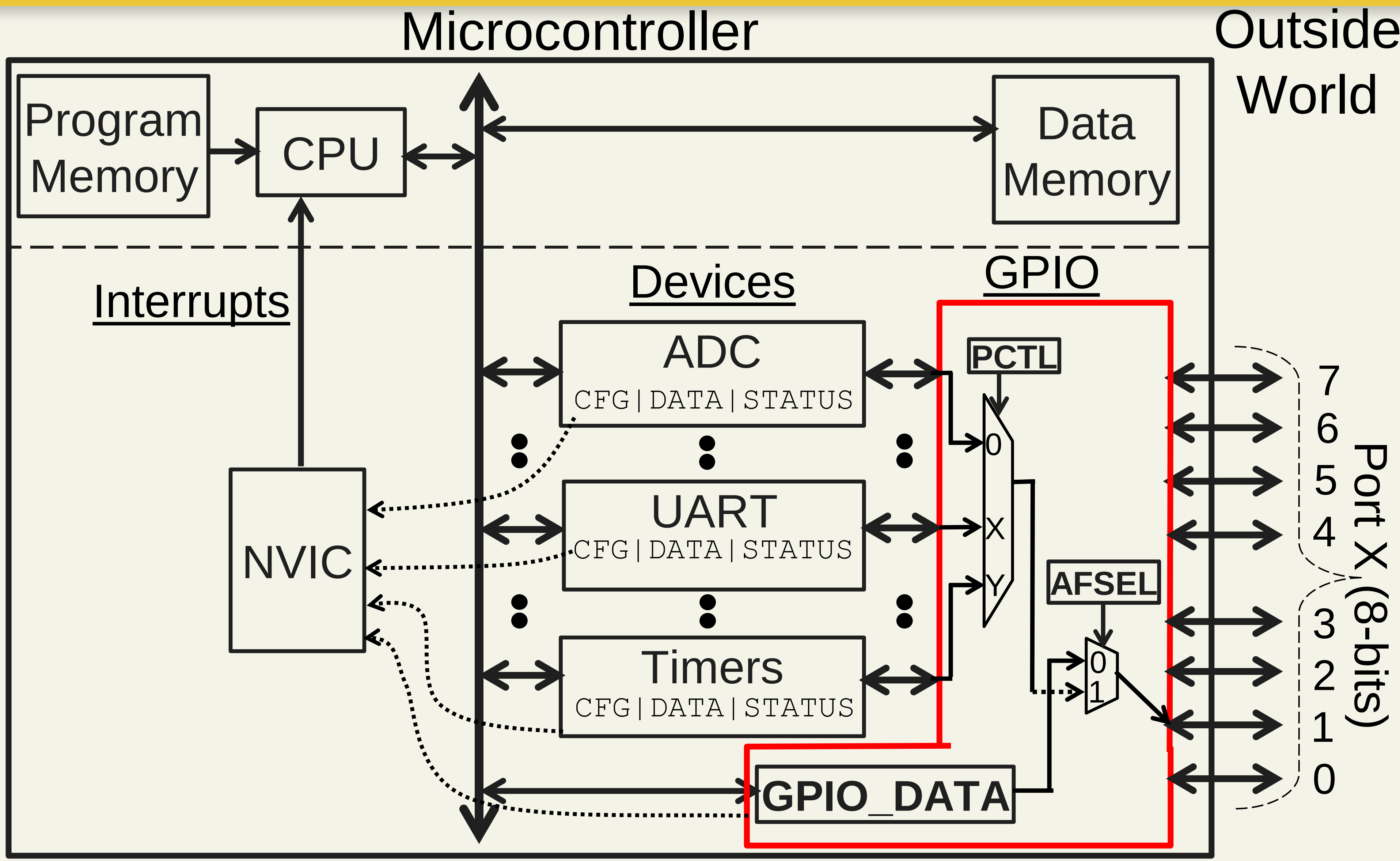


Figure 2.17 Function block diagram of Analog/Digital GPIO control.

Microcontroller / System-on-Chip (SoC)



GPIO Port: Alternative Function Architecture

- GPIOAFSEL: For each wire select GPIO or Alternative Function
- GPIOPCTL: For each wire choose which Alternative Function

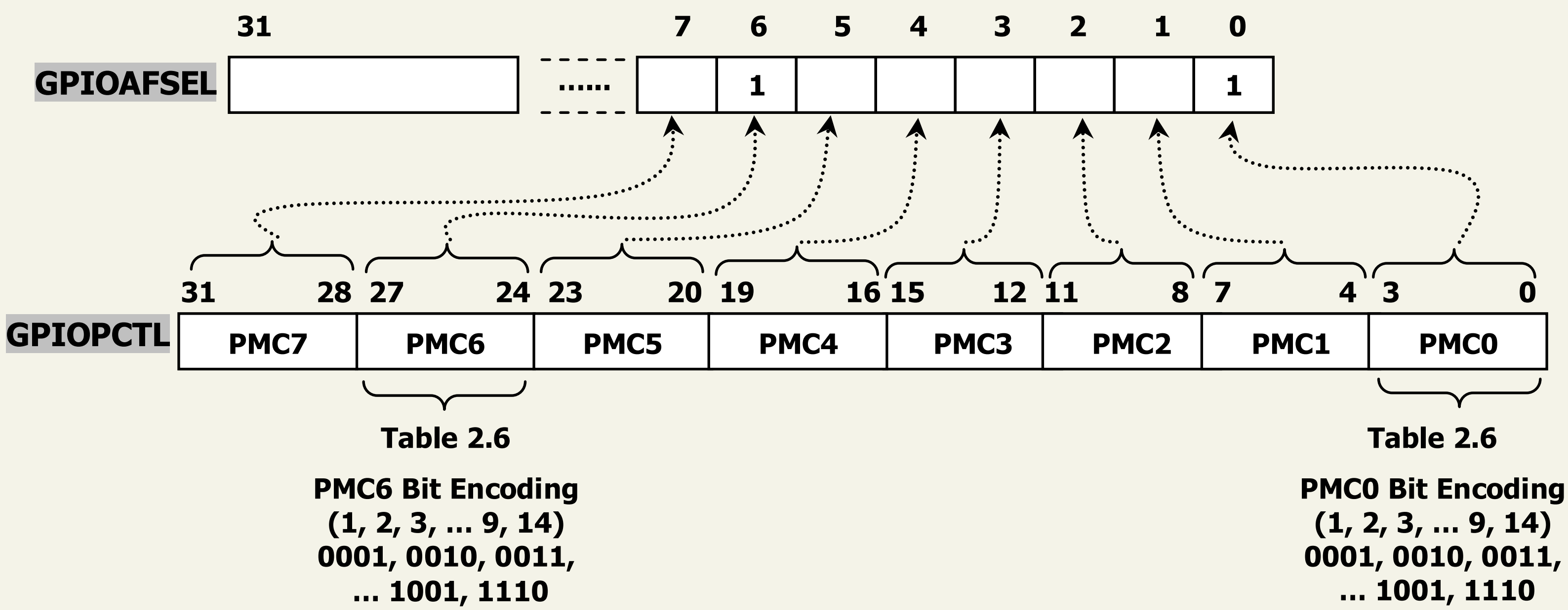


Figure 2.19 The illustration for GPIOAFSEL and GPIOPCTL registers.

GPIO Port: Alternative Function Architecture

Table 2.6 GPIO Pins and Alternate Functions

I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)									
			1	2	3	4	5	6	7	8	9	14
PA0	17	-	U0RX	-	-	-	-	-	-	CAN1RX	-	-
PA1	18	-	U0TX	-	-	-	-	-	-	CAN1TX	-	-
PA2	19	-	-	SSI0CLK	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0FSS	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0RX	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0TX	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-
PA7	24	-	-	-	I2C1SDC	-	M1PWM3	-	-	-	-	-
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-	-	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-	-	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX	-	-
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX	-	-
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-	-	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-	-	-
PC0	52	-	TCK SWCLK	-	-	-	-	-	T4CCP0	-	-	-
PC1	51	-	TMS SWDIO	-	-	-	-	-	T4CCP1	-	-	-
PC2	50	-	TDI	-	-	-	-	-	T5CCP0	-	-	-
PC3	49	-	TDO SWO	-	-	-	-	-	T5CCP1	-	-	-
PC4	16	C1-	U4RX	U1RX	-	M0PWM6	-	IDX1	WT0CCP0	U1RTS	-	-
PC5	15	C1+	U4TX	U1TX	-	M0PWM7	-	PHA1	WT0CCP1	U1CTS	-	-

GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure Port B to use UART1 TX?**

GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure <u>Port B</u> to use <u>UART1 TX</u>?**

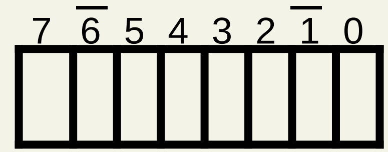
I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)							
			1	2	3	4	5	6	7	8
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-

GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure Port B to use UART1 TX?**

I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)							
			1	2	3	4	5	6	7	8
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-

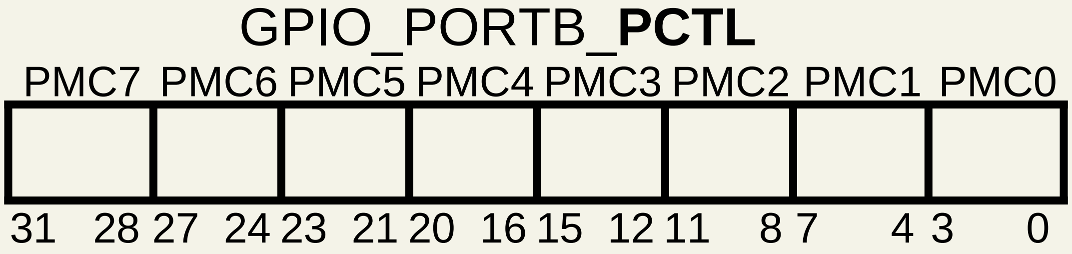
GPIO_PORTB_**AFSEL**

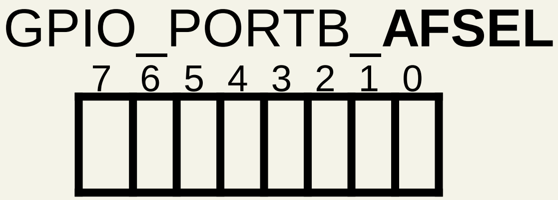


GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure Port B to use UART1 TX?**

I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)							
			1	2	3	4	5	6	7	8
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-

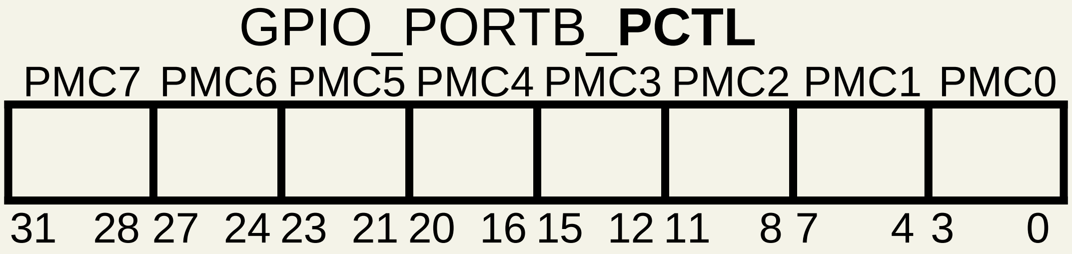


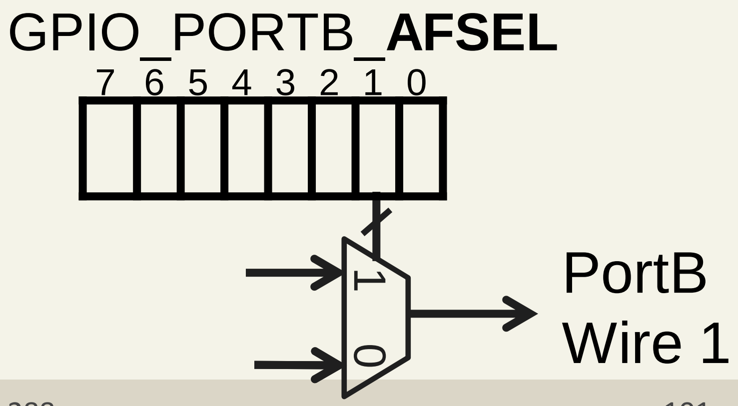


GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure Port B to use UART1 TX?**

I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)							
			1	2	3	4	5	6	7	8
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-





GPIO Port: Alternative Function Architecture

- **What values do GPIOAFSEL and GPIOPCTL need to be given to configure Port B to use UART1 TX?**

I/O	Pin	Analog Function	Digital Functions (GPIOPCTL PMCx Bit Field Encoding)							
			1	2	3	4	5	6	7	8
PB0	45	USB0ID	U1RX	-	-	-	-	-	T2CCP0	-
PB1	46	USB0VBUS	U1TX	-	-	-	-	-	T2CCP1	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-
PB3	48	-	-	-	I2C0SDC	-	-	-	T3CCP1	-
PB4	58	AIN10	-	SSI2CLK	-	M0PWM2	-	-	T1CCP0	CAN0RX
PB5	57	AIN11	-	SSI2FSS	-	M0PWM3	-	-	T1CCP1	CAN0TX
PB6	1	-	-	SSI2RX	-	M0PWM0	-	-	T0CCP0	-
PB7	4	-	-	SSI2TX	-	M0PWM1	-	-	T0CCP1	-