M01: Embedded System Architecture

1.4. Basic programming

Prof. Rosa Zheng

Ref: www.geeksforgeeks.org

Alex Allain, www.cprogramming.com

J. Valvano, Embedded Systems: Shape the World.

The C Language Standards

- C99 is used, as mentioned in the Tiva C workshop instructions.
- C99 is a C standard published in 1999, i.e. ISO/IEC9899:1999 by ITU (International Telecommunication Union).
- C11: the 2011 version;
- The latest C standard is ISO/IEC 9899:2018, also known as C17 / C18.

(https://www.iso.org/standard/74528.html)

C Programming Tutorial

- Lots of websites. Good ones:
- Geeks for Geeks: https://www.geeksforgeeks.org/c-programming-language/
  - different languages and interesting topics
  - Lots of examples, can run by online IDE
- Alex Allain: www.cprogramming.com
  - Topics explained well for beginners
  - Good reference value too.

Unspecified Behaviors

- **The order in which the function designator, arguments, and subexpressions within the arguments are evaluated in a function call.**
- **Example: https://www.geeksforgeeks.org/c-programming-language-standard/ (run on IDE)**

```
#include<stdio.h>
int main()
{
    int i = 1;
    printf("%d %d %d\n", i++, i++, i);
    return 0;
}
```

Results? It depends.

2 1 3

- using g++ 4.2.1 on Linux.i686

1 2 3

- using SunStudio C++ 5.9 on Linux.i686

What is the solution?

- **Avoid the undefined or unspecified programming constructs.**
- **Never change state within an expression**

```
#include<stdio.h>
int main()
{
    int i = 1;
    int j= i+1;
    int k= j+1;
    printf("%d %d %d\n", i, j, k);
    return 0;
}
```

Use ++i or i++ all by itself with "no surrounding expression". The two loops below behave the same:

```
for(i = 0; i < 10; ++i)
        printf("%d\n", i);

for(i = 0; i < 10; i++)
        printf("%d\n", i);
```

The 90-90 Rule

- The 80-20 rule (Pareto principle or the law of the vital few):

For many events, roughly 80% of the effects come from 20% of the causes.

- Tom Cargill of Bell Labs said in the 1980s:

The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time.

code: 90%+10%=100%, time: 90% + 90% = 180%

What good codes look like?

- Correct – does what it supposed to do
- Readable – formatting, naming, commenting,
- Simple – the KISS principle
- Maintainable
  - Rugged - difficult to misuse, kind to errors
  - Reliable - runs under different conditions
  - Extendable – easy to add other modules / functions
- Efficient - fast enough for its designed functionality

http://www.maultech.com/chrislott/resources/cstyle/

Best Coding Practices

- Always write in American English: naming conventions, comments, etc.
- Always write to the latest standard and recommended style;
- Be consistent in naming and formatting.
  - Naming what, not how;
  - Use space instead of tab for indentation;
  - Never exceed 79 characters per line
- Provide functional comments
- Move loop-invariants outside of loop

C Preprocessors

- Directives (where to find the files):
  - #include <stdint.h>
  - #include “driverlib/sysctl.h”
- Constants (note the brackets):
  - Example: #define PI_PLUS_ONE (3.14 + 1)
- Macros (note the brackets):
  - #define MULT(x, y) ((x) * (y))
- Conditional compilation:
  - #if … #elif … #else … #endif
  - #ifdef … #endif or #ifndef … #endif

Pointer Variables

- **Pointer syntax:**
  - Declaration: int *p1, non_pointer; or int *p1, *p2;
  - Assignment: int x; p=&x;
  - Value contained in pointer p is *p;
- **Example: run this @ the online IDE**

```
#include <stdio.h>
int main()
{
   int x;          /* A normal integer*/
   int *p;         /* A pointer to an integer ("*p" is an integer, so p
                   must be a pointer to an integer) */

   p = &x;         /* Read it, "assign the address of x to p" */
   scanf( "%d", &x );        /* Put a value in x, we could also use p here */
   printf( "%d\n", *p ); /* Note the use of the * to get the value */
   getchar();
}
```

TivaWare driverlib User Guide

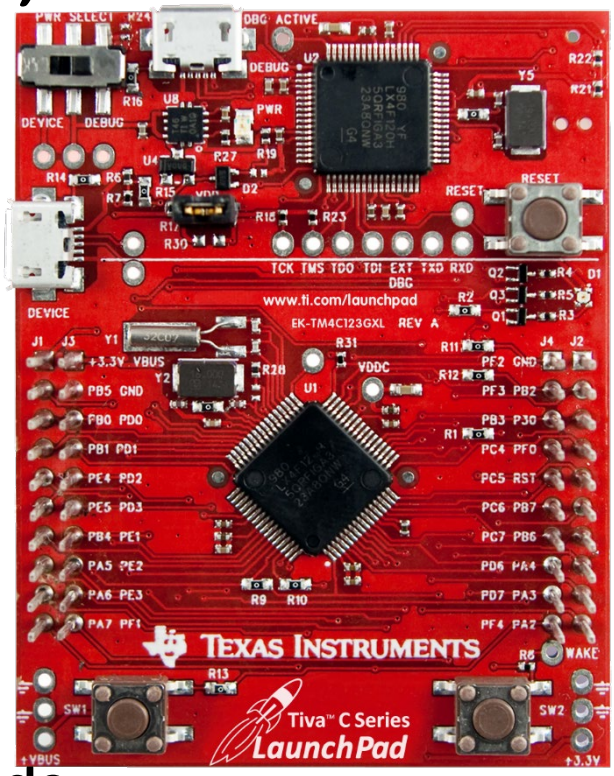
- Two programming models:
  - Direct register access model:
    - Access registers directly, high performance
    - Macros defined in header file inc/ tm4c123gh6pm.h or hw_*.h, don't use both
    - Naming conventions: xxx_yyy_suffix,
    - Suffix: _R, _M, _S, _bitField, except GPIO: Px0
    - Names in xxx, yyy, bitField match datasheet
  - Software driver model (APIs)
  - Two models can be used together

Analyze Lab 2 files

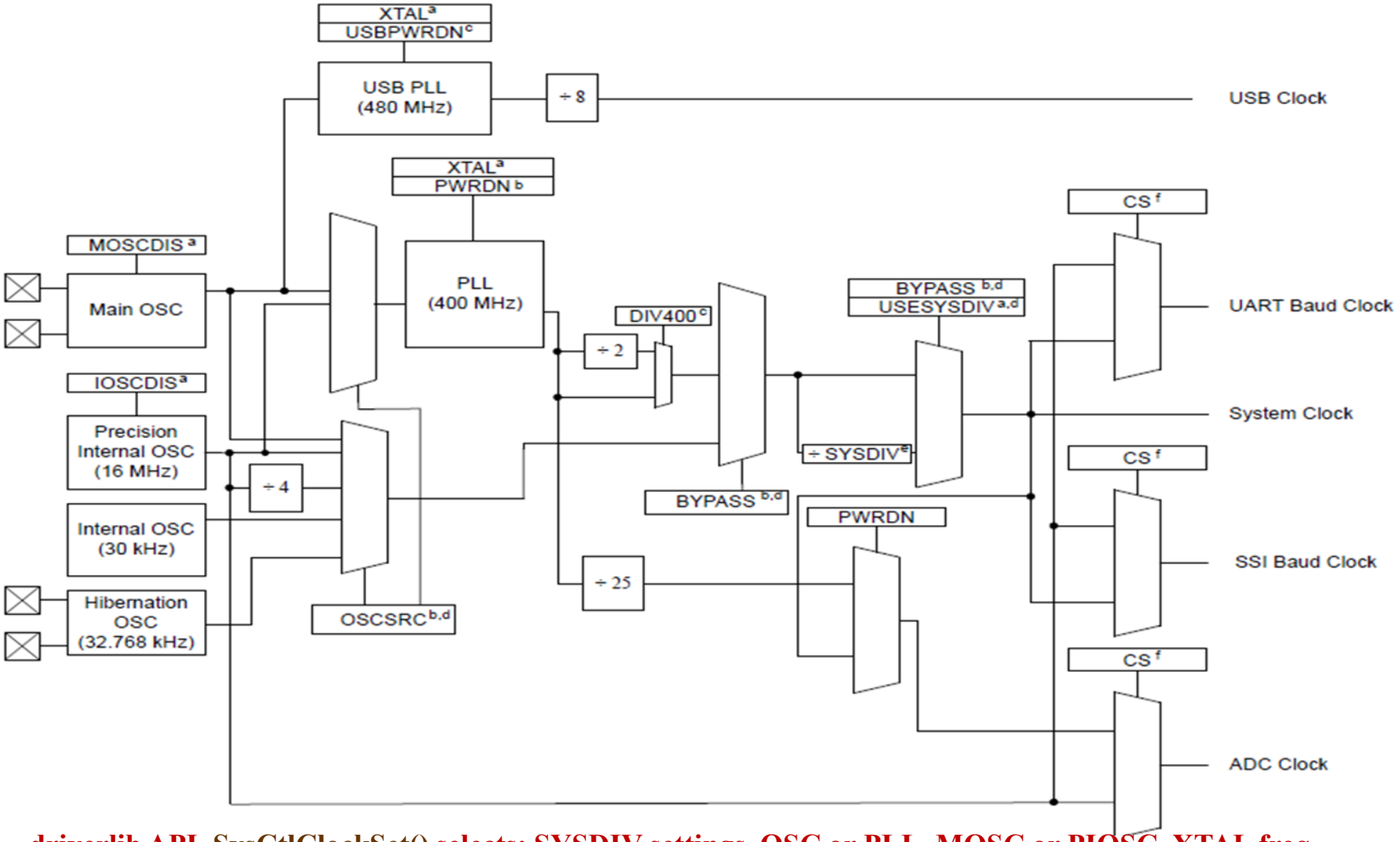
- What are the include files in main.c?
  - Look into sysctl.h and hw_types.h, see what are in there.
  - Read C preprocessors tutorial page on cprogramming.com and get familiar with the four types of preprocessors: directives, constants, conditional compilation, and macros.
- What are the APIs used in main.c?
  - Read the related chapters in driverlib user guide and understand how the APIs work.

System Clocks

- **Precision Internal Oscillator (PIOSC)**
  - 16 MHz ± 3% (1% w/ calibration)
  - Used after POR (Power-on reset)
- **Main Oscillator (MOSC) using…**
  - An external single-ended clock source
  - An external crystal – on board 16 MHz
- **Internal 30 kHz Oscillator (LFIOSC)**
  - 30 kHz ± 50%, Intended for use during Deep-Sleep power-saving modes
- **Hibernation Module Clock Source**
  - 32,768 Hz crystal – on board Y1
  - a real-time clock source and accurate source for Deep-Sleep or Hibernate mode
- **Details in data sheet (clock control)**

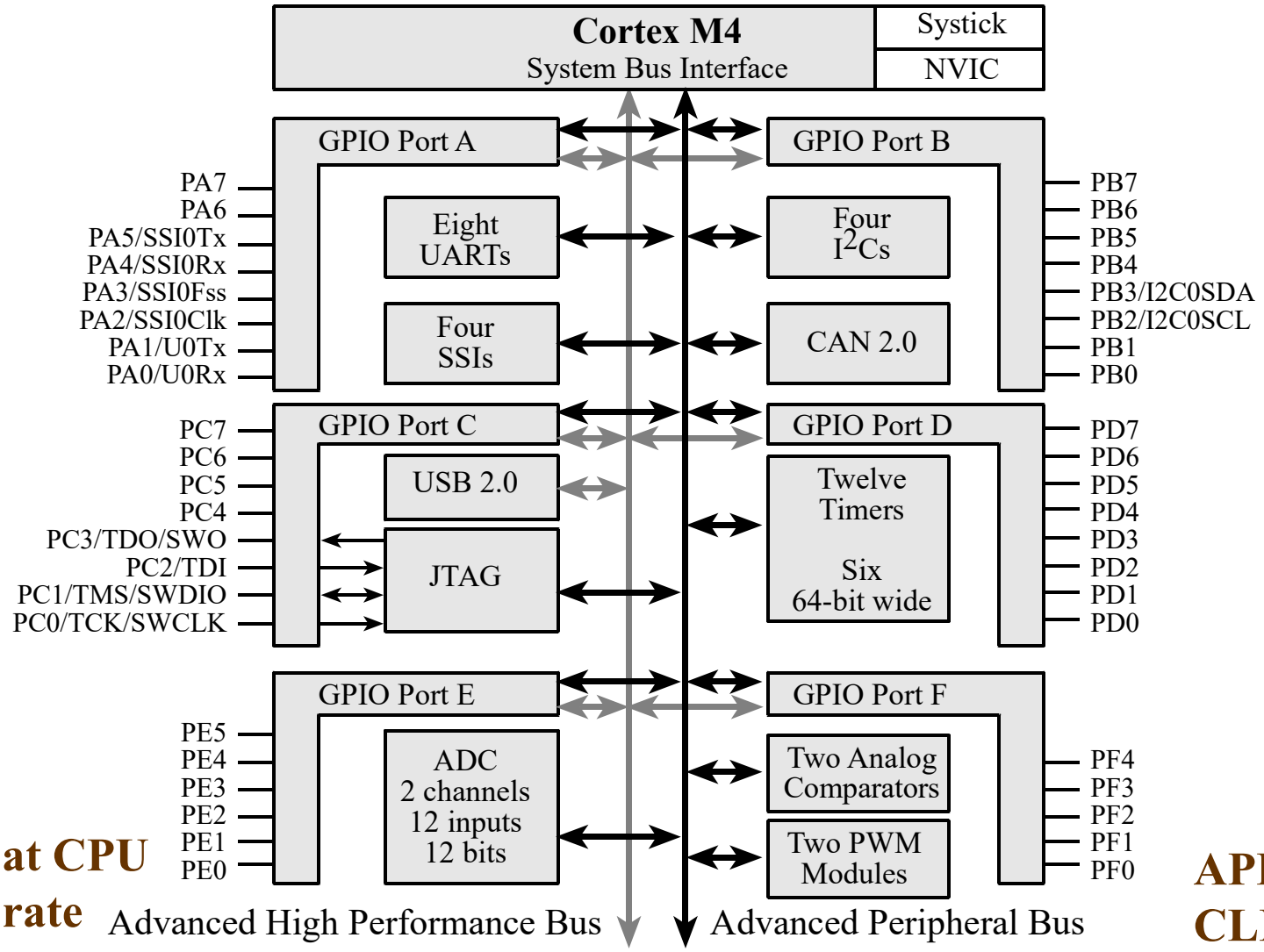


Tiva C Series Clock Tree



driverlib API SysCtlClockSet() selects: SYSDIV settings, OSC or PLL, MOSC or PIOSC, XTAL freq

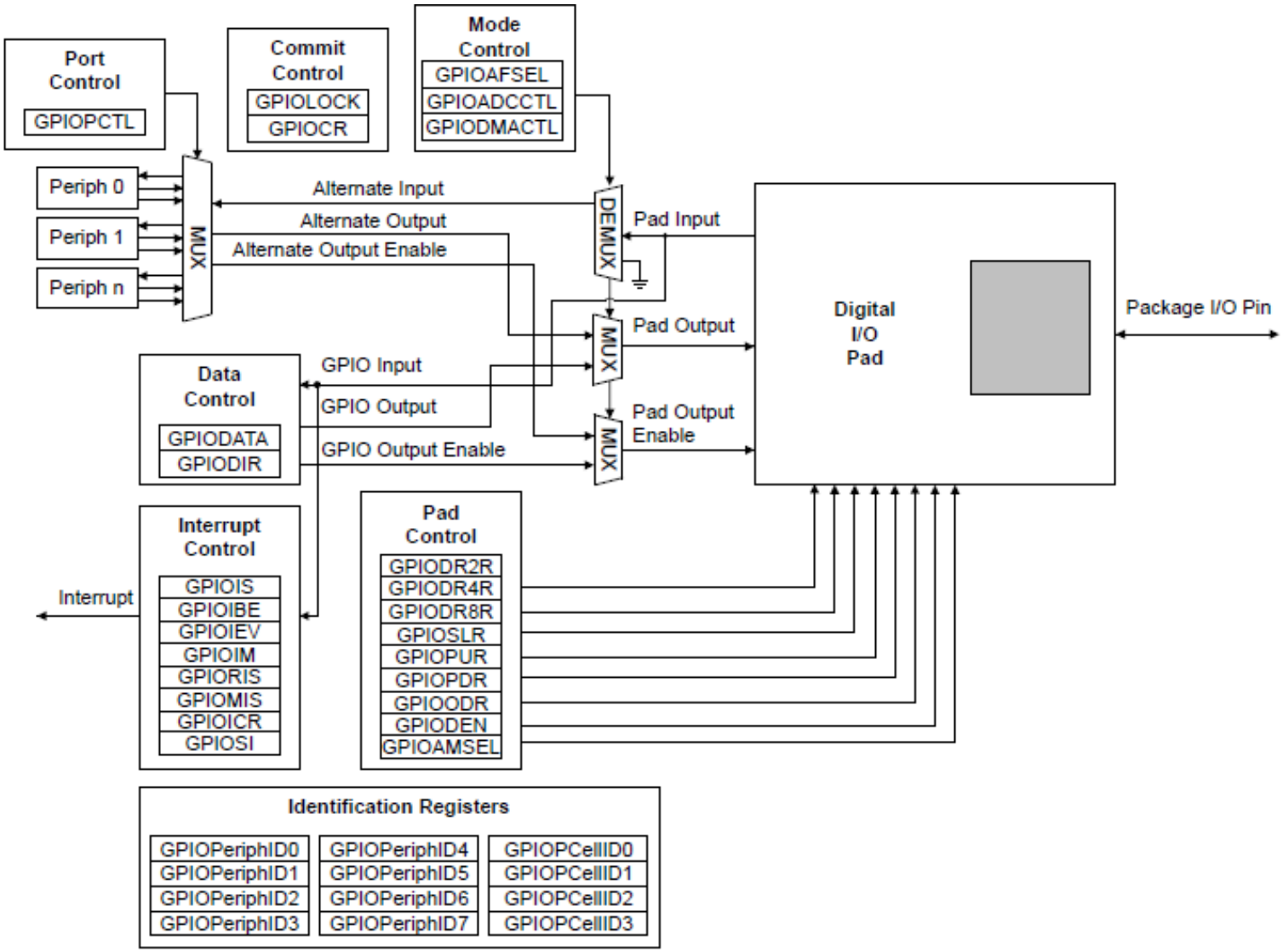
General Purpose IO



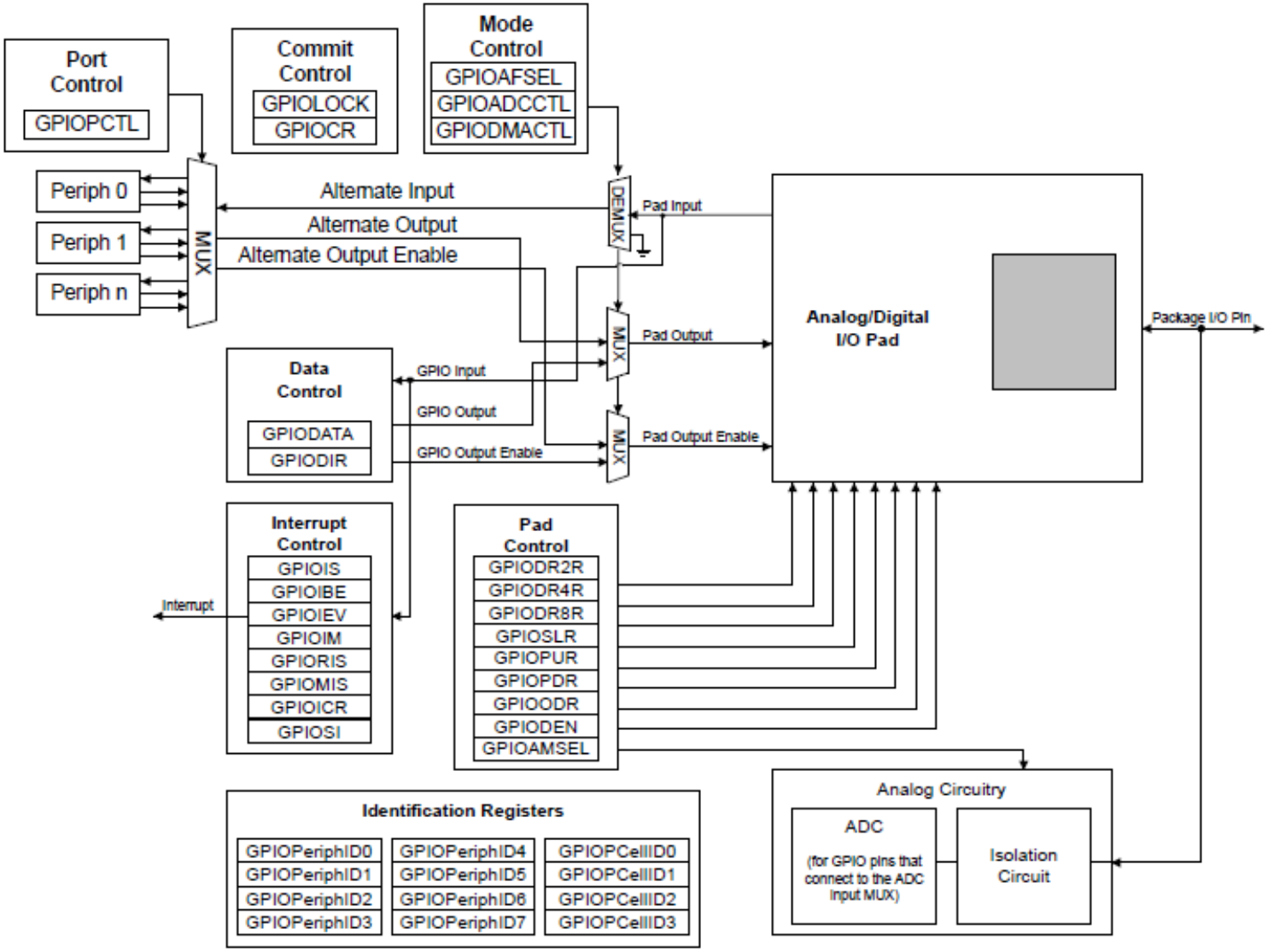
AHB at CPU CLK rate

APB at ½ CPU CLK rate

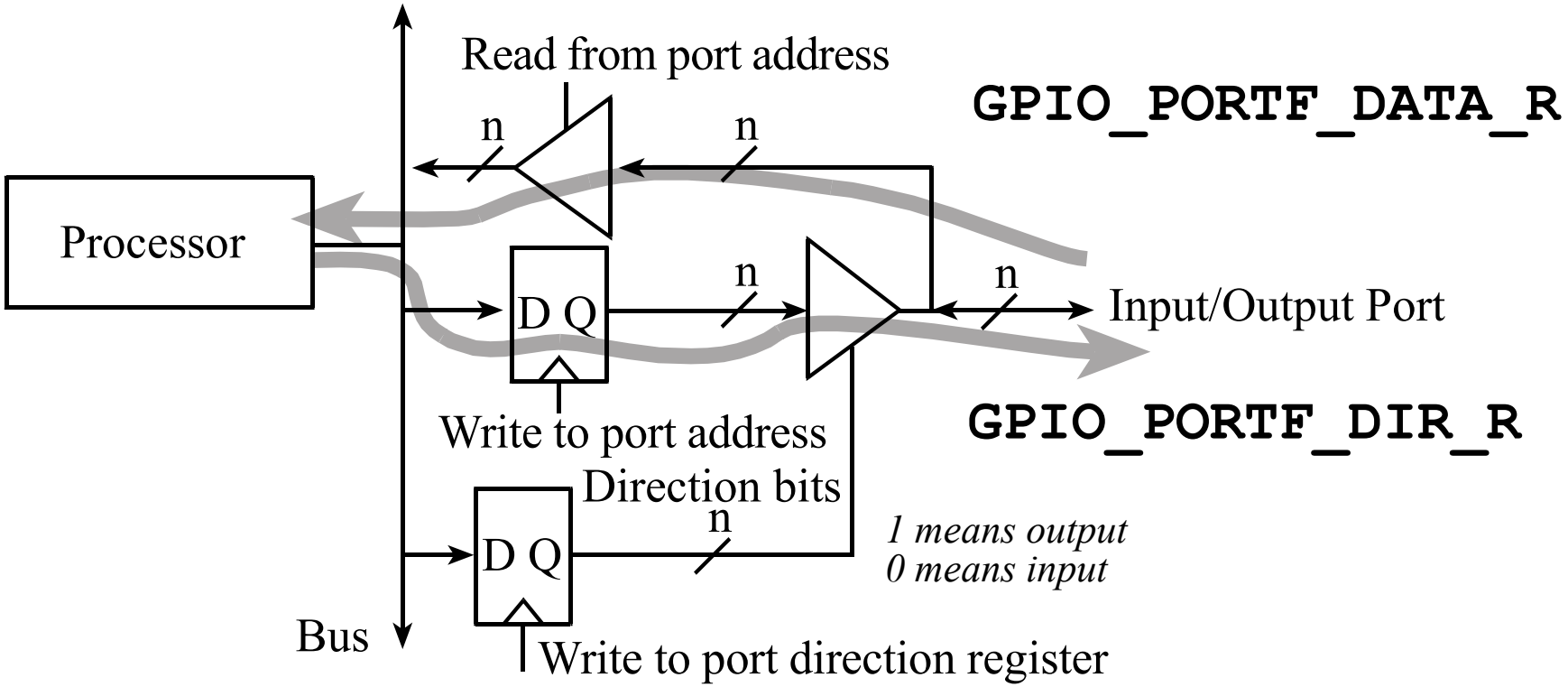
Digital GPIO Pads



Analog/Digital GPIO Pads



Programmable Input / Output



- **Drive Strength**
  - **Programmable 2 mA, 4 mA, 8mA or 8mA w/ slew rate control**
  - **Check the LED drive circuits of the LaunchPad**
- **Sink Capacity: Up to four pads with 18 mA input current**

GPIO Address Masking

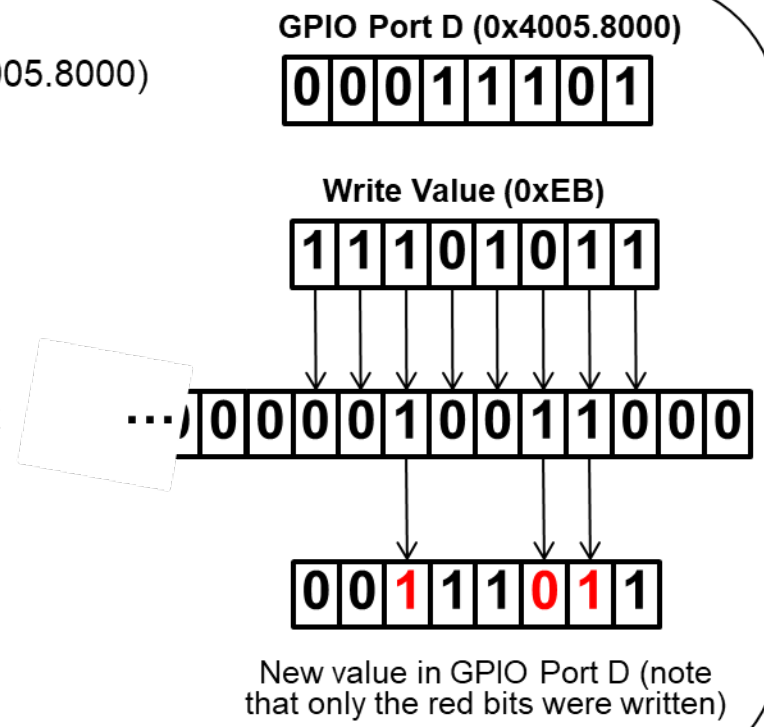
GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_5|GPIO_PIN_2|GPIO_PIN_1, 0xEB);

The register we want to change is GPIO Port D (0x4005.8000)
Current contents of the register is:

The value we will write is 0xEB:

Instead of writing to GPIO Port D directly, write to 0x4005.8098. Bits 9:2 (shown here) become a bit-mask for the value you write.

Only the bits marked as "1" in the bit-mask are changed.



Note: you specify base address, bit mask, and value to write.
The GIPOPinWrite() function determines the correct address for the mask.

Critical Function GPIO Protection

- Six pins on the device are protected against accidental programming:
  - PC3,2,1 & 0: JTAG/SWD
  - PD7 & PF0: NMI
- Any write to the following registers for these pins will not be stored unless the GPIOLOCK register has been unlocked:
  - GPIO Alternate Function Select register
  - GPIO Pull Up or Pull Down select registers
  - GPIO Digital Enable register
- The following sequence will unlock the GPIOLOCK register for PF0 using direct register programming:

```
HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
```

- Reading the GPIOLOCK register returns it to lock status