M3 - GPIO

GPIO Configuration and Pins

For general purpose ports

Interrupts review

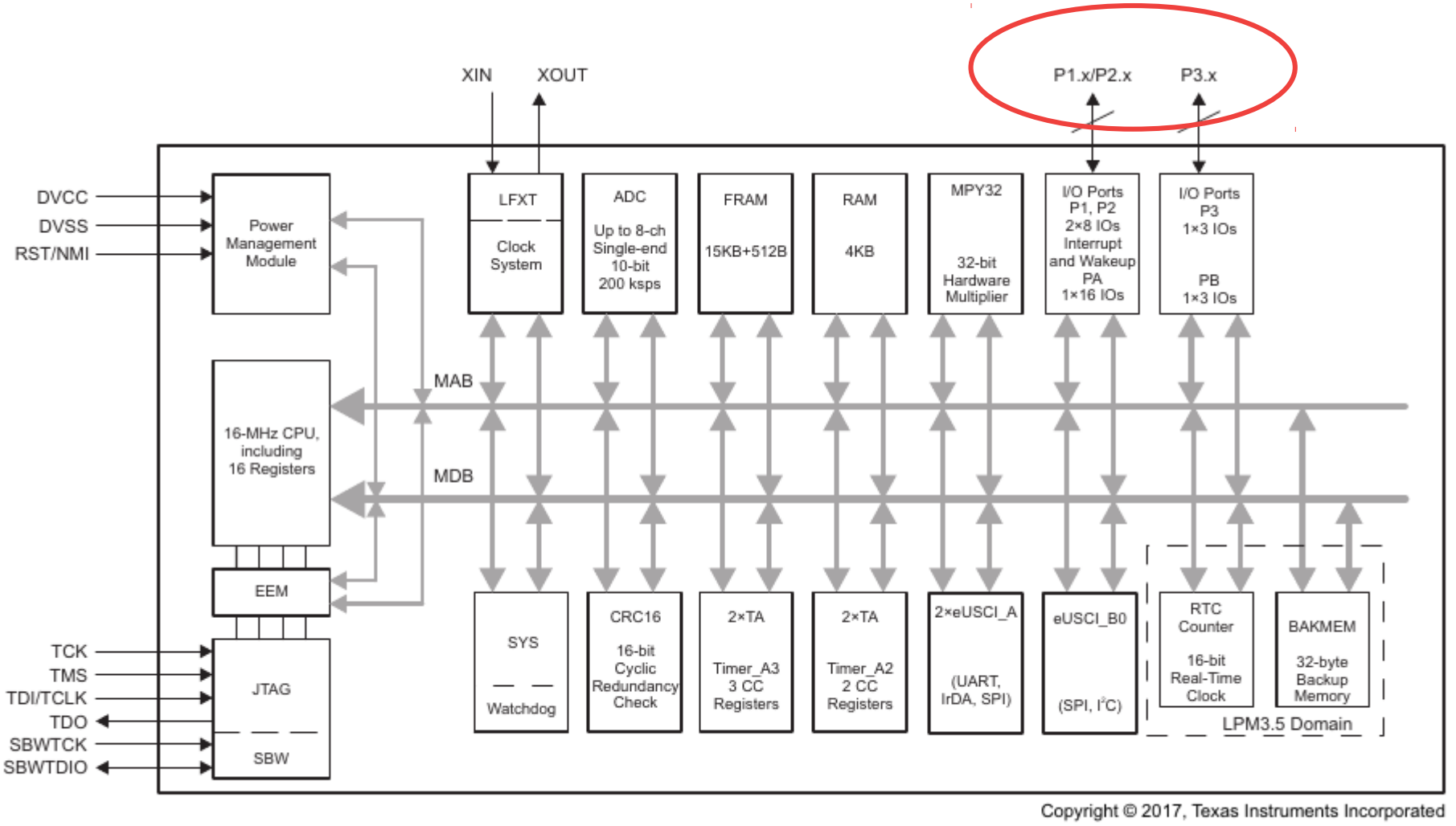
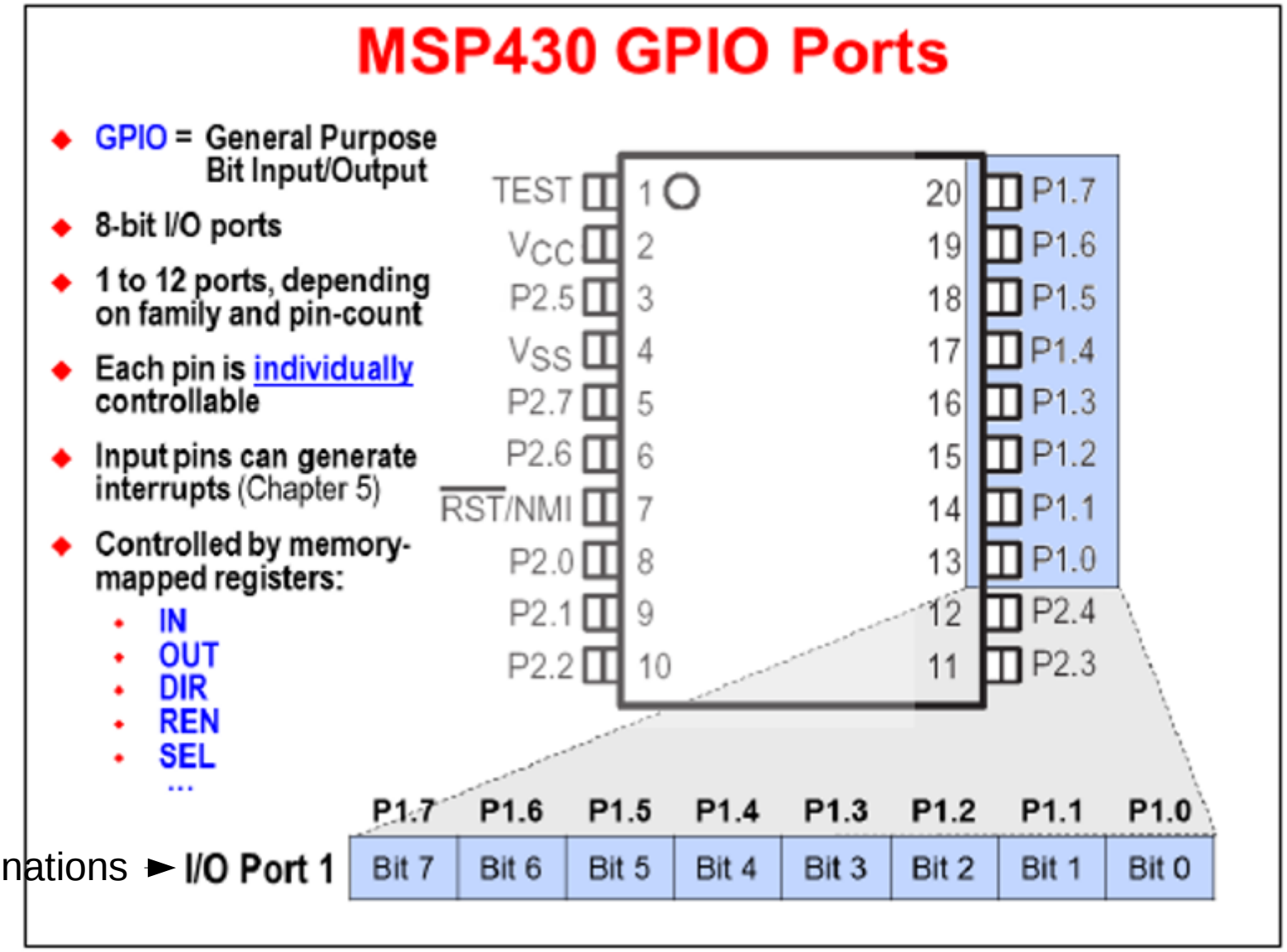


Figure 1-1. Functional Block Diagram

The MSP430 provides one or more 8-bit I/O ports. The number of ports is often correlated to the number of pins on the device – more pins, more I/O. The I/O port bits (and their related pins) are enumerated with a Port number, along with the bit/pin number; for example, the first pin of Port 1 called: P1.0.



MSP430FR2433 Launchpad Pinouts
Functions for pins are multiplexed, the arrows point to the Port.Pin designation
For the connections to the LaunchPad Board

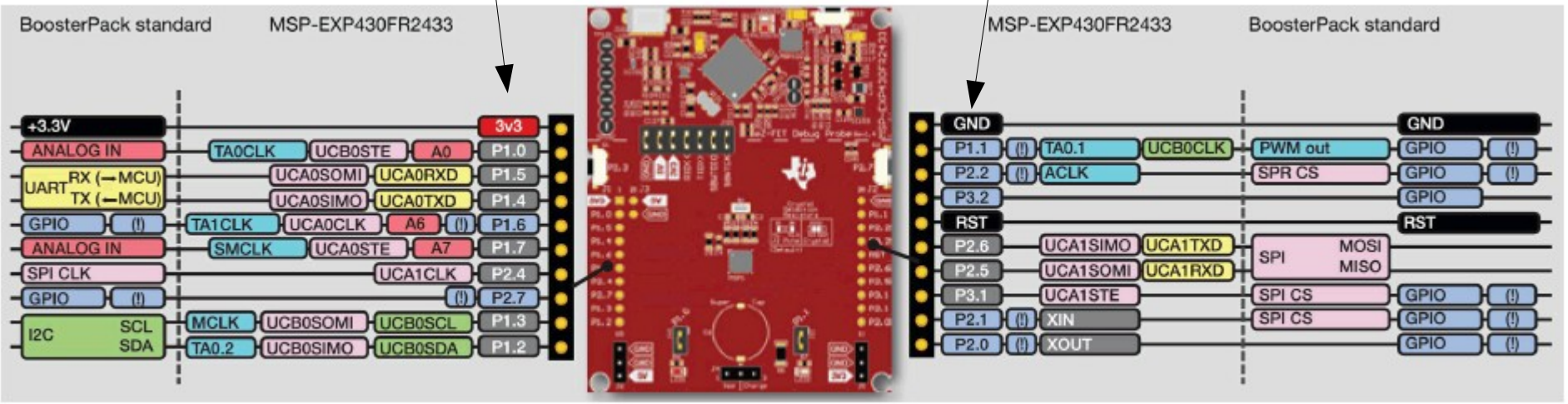
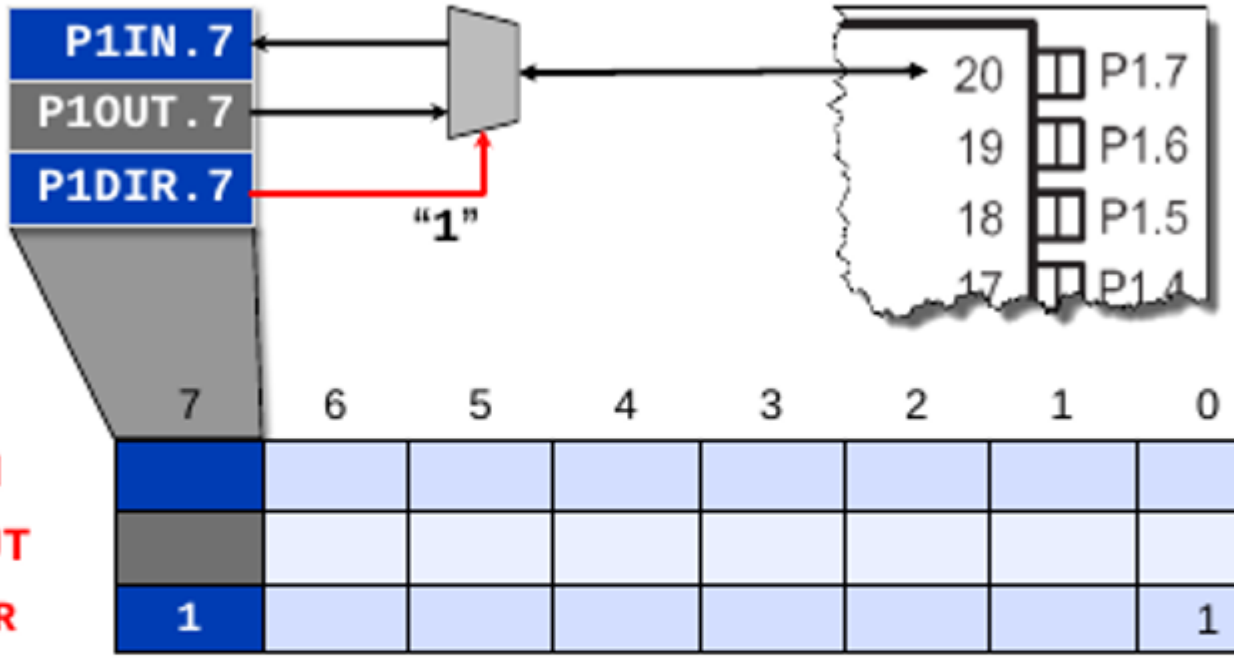


Figure 11. LaunchPad Kit to BoosterPack Module Connector Pinout

PxDIR (Pin Direction): Input or Output

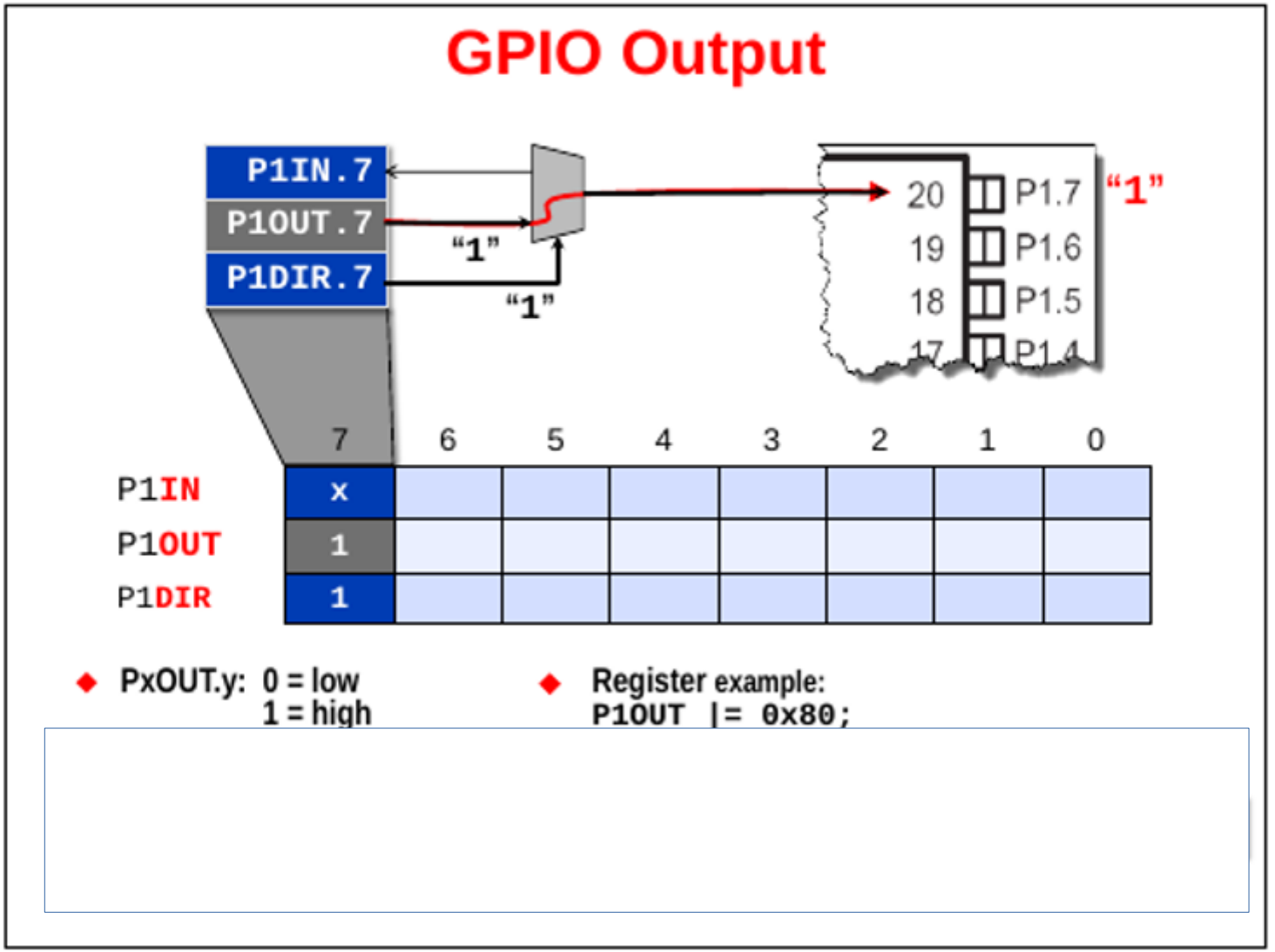


	7	6	5	4	3	2	1	0
P1IN								
P1OUT								
P1DIR	1							1

- PxDIR.y: 0 = input
  1 = output
- Register example:
  `P1DIR |= 0x81;`

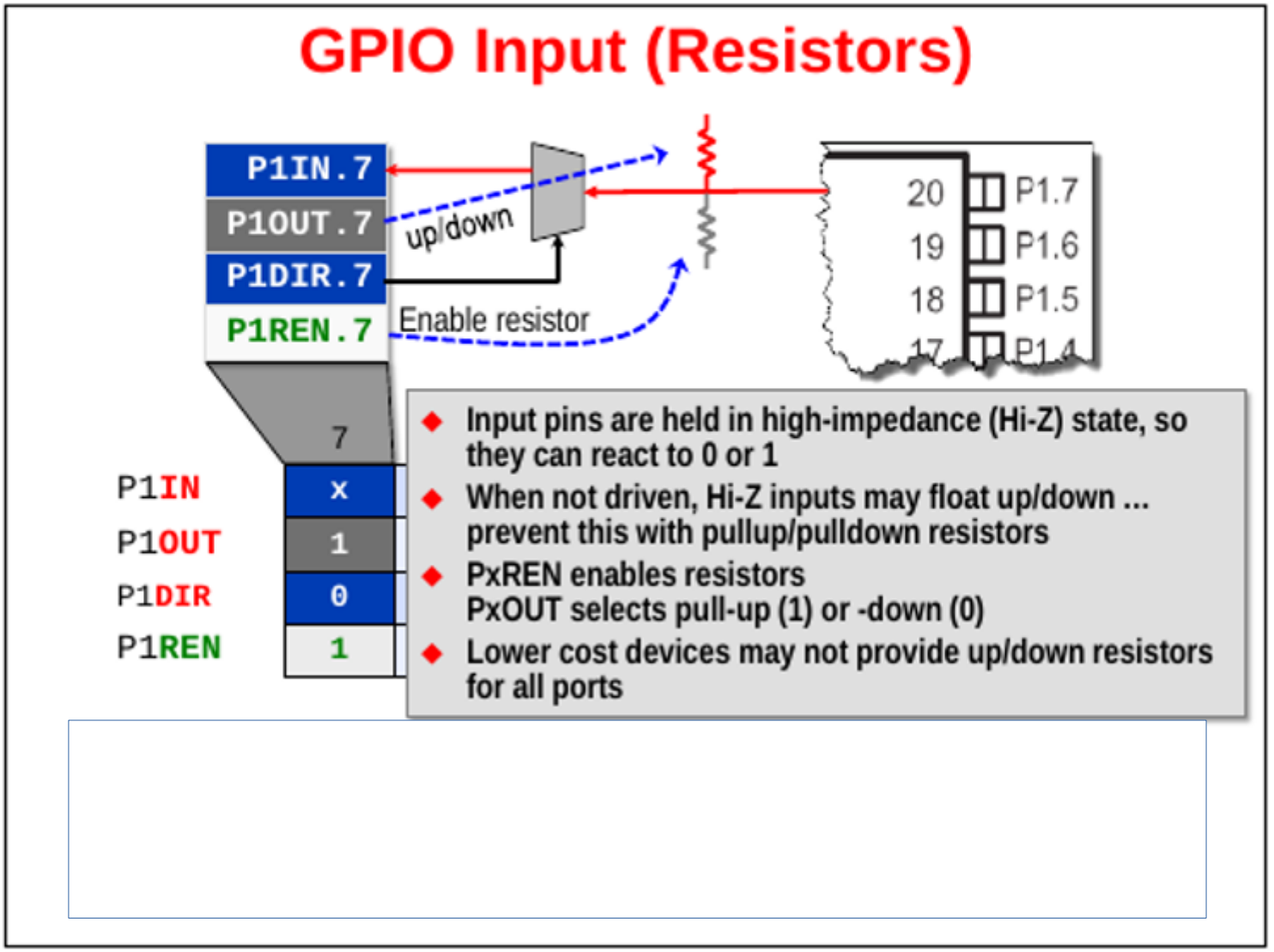
GPIO Output

Once you've configured a pin as an output with the PxDIR register, you can set the pins value using the PxOUT register. For P1.7, this would be the P1OUT register.

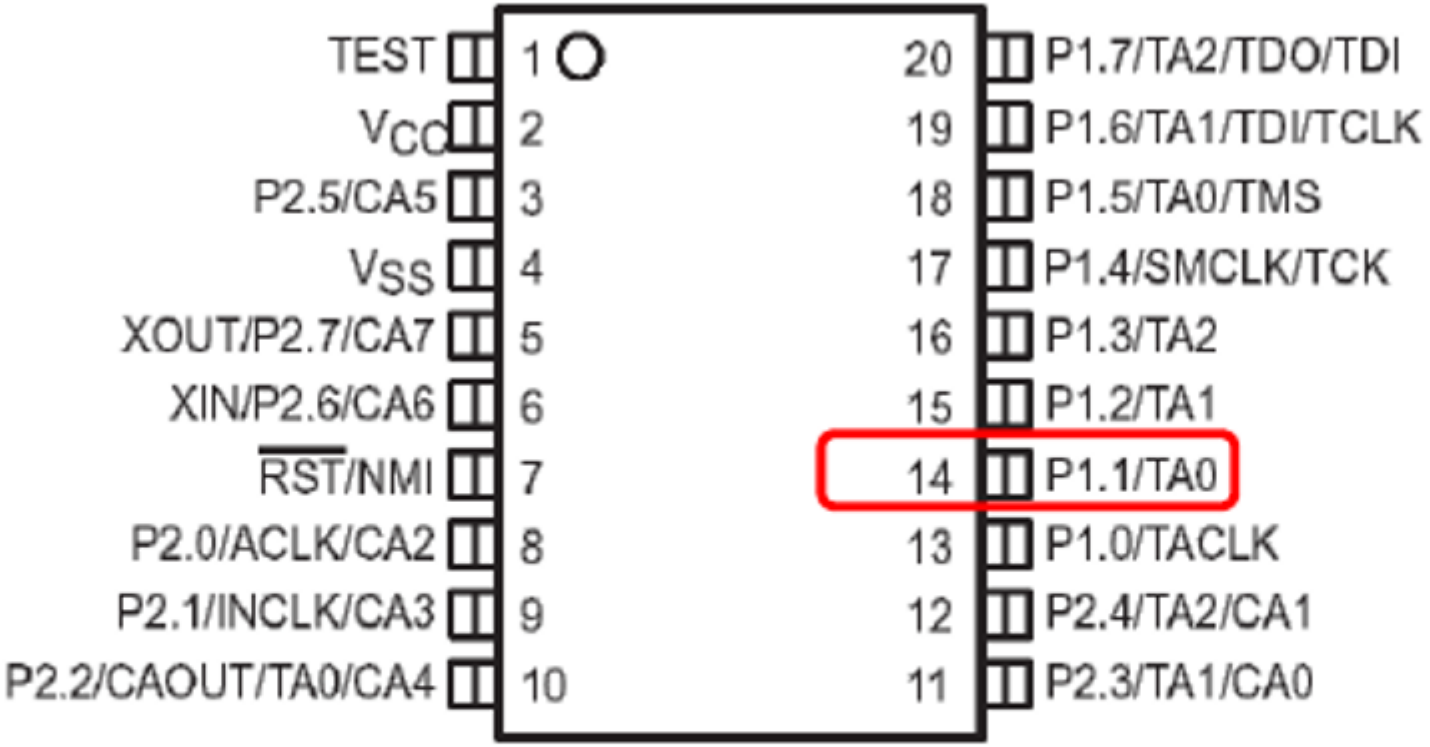


GPIO Input

Reading a pin's value is done by reading the PxIN register. The `GPIO_getInputValue()` Driver function returns this value to a variable in your program.

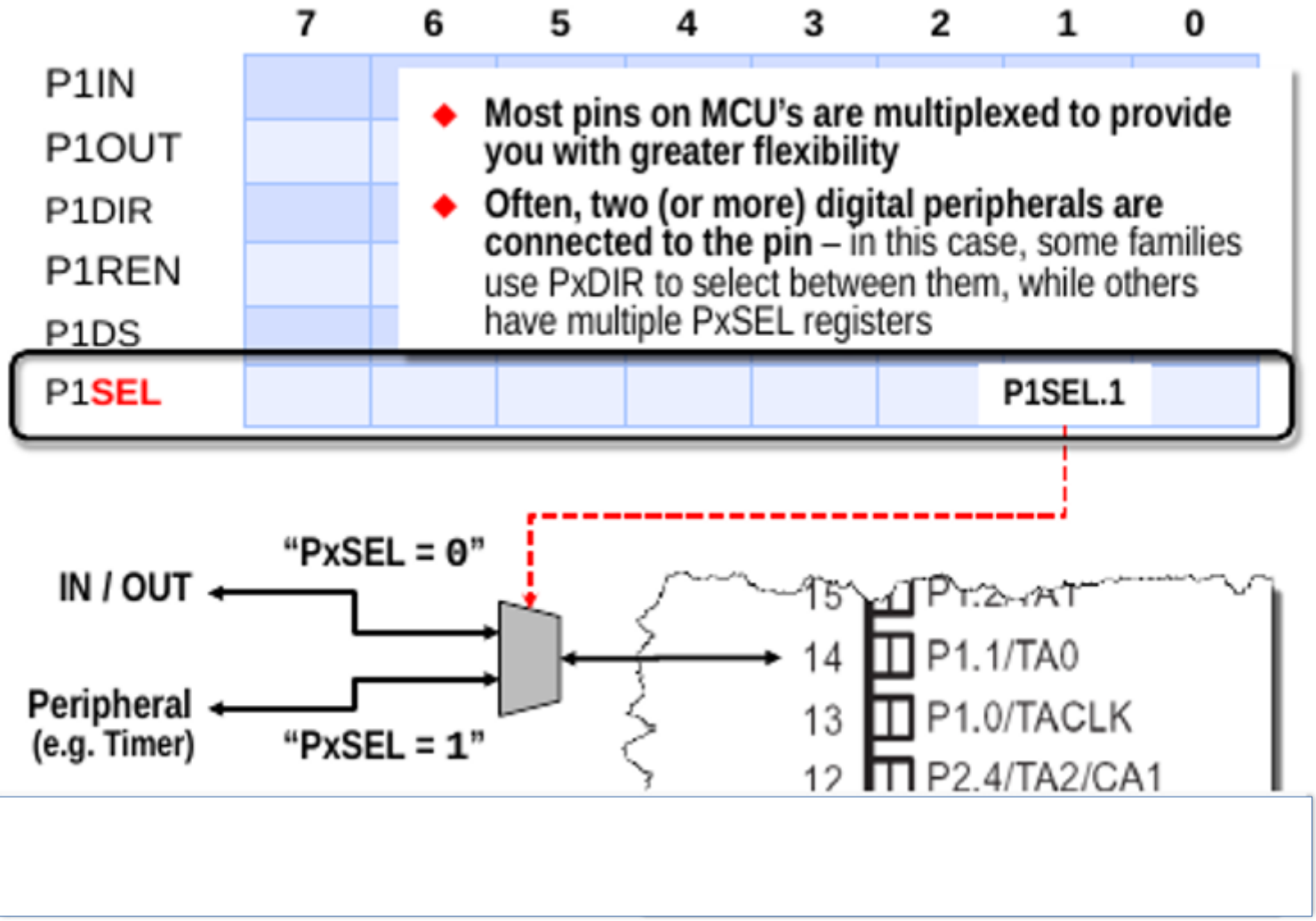


Controlling GPIO Ports



- Most pins on MCU's are multiplexed to provide you with greater flexibility – which peripherals do you want to use in your system

Pin Flexibility



Devices with Multiple Pin Selection Registers

Some MSP430 devices actually have two *pin select* registers, as this is needed to support the gre
number of pin mux options.

P1SEL0 & P2SEL1: FR5969 Example

Table 50. Port P1 (P1.0 to P1.2) Pin Functions (From the 'FR5969 datasheet)

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS/SIGNALS[1]		
			P1DIR.x	P1SEL1.x	P1SEL0.x
P1.0/TA0.1/DMAE0/RTCCLK/A0/C0/VREF-/VeREF-	0	P1.0 (I/O)	I: 0; O: 1	0	0
		TA0.CCI1A	0	0	1
		TA0.1	1		
		DMAE0	0	1	0
		RTCCLK	1		

Da

Table 8-2. I/O Function Selection (From the 'FR5969 User's Guide)

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

- The User's Guide tells us how to read the datasheet
- DriverLib uses *I/O Function* name

```
// Set pin P1.0 to output TA0.1 (which is the CCR1 output signal for TIMER0_A)
GPIO_setAsPeripheralModuleFunctionOutputPin(
        GPIO_PORT_P1,                       // I/O Port number
        GPIO_PIN0,                          // Pin Number
        GPIO_PRIMARY_MODULE_FUNCTION );     // Which peripheral function
                                            //   (primary, secondary, ternary)
```

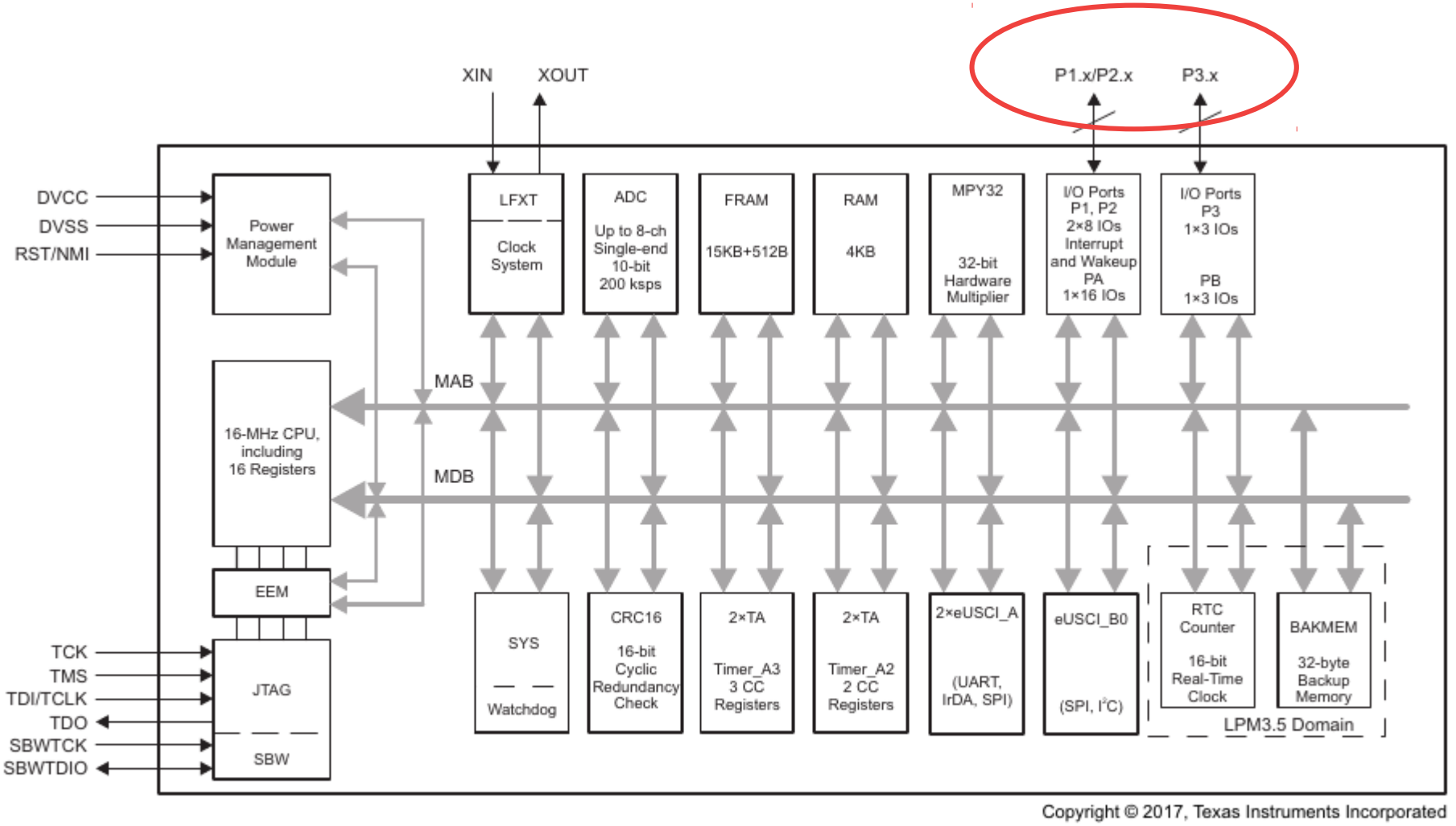


Figure 1-1. Functional Block Diagram

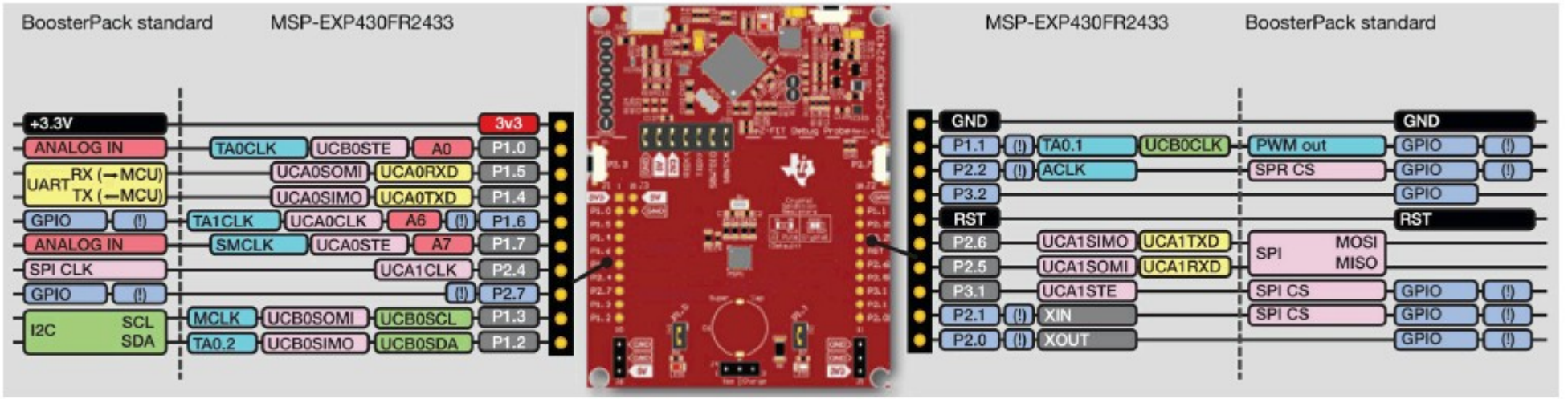


Figure 11. LaunchPad Kit to BoosterPack Module Connector Pinout

Digital I/O

This chapter describes the operation of the digital I/O ports in all devices.

Topic **Page**

7.2 Digital I/O Operation

The digital I/Os are configured with user software. The setup and operation of the digital I/Os are discussed in the following sections.

7.2.1 Input Registers (PxIN)

Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function. These registers are read only.

- Bit = 0: Input is low
- Bit = 1: Input is high

NOTE: Writing to read-only registers PxIN

Writing to these read-only registers results in increased current consumption while the write attempt is active.

7.2.2 Output Registers (PxOUT)

Each bit in each PxOUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction.

- Bit = 0: Output is low
- Bit = 1: Output is high

If the pin is configured as I/O function, input direction and the pullup or pulldown resistor are enabled; the corresponding bit in the PxOUT register selects pullup or pulldown.

- Bit = 0: Pin is pulled down
- Bit = 1: Pin is pulled up

7.2.3 Direction Registers (PxDIR)

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

- Bit = 0: Port pin is switched to input direction
- Bit = 1: Port pin is switched to output direction

7.2.4 Pullup or Pulldown Resistor Enable Registers (PxREN)

Each bit in each PxREN register enables or disables the pullup or pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin contains a pullup or pulldown.

- Bit = 0: Pullup or pulldown resistor disabled
- Bit = 1: Pullup or pulldown resistor enabled

Table 7-1 summarizes the use of PxDIR, PxREN, and PxOUT for proper I/O configuration.

Table 7-1. I/O Configuration

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

7.2.5 Function Select Registers (PxSEL0, PxSEL1)

Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each port pin uses two bits to select the pin function: I/O port or one of the three possible peripheral module functions. Table 7-3 shows how to select the various module functions. See the device-specific data sheet to determine pin functions. Each PxSEL bit is used to select the pin function: I/O port or peripheral module function. A device in this family may have only PxSEL0 or both PxSEL0 and PxSEL1.

Table 7-2. I/O Function Selection for Devices With Only One Selection Bit – PxSEL0

PxSEL0	I/O Function
0	General purpose I/O is selected
1	Primary module function is selected

Table 7-3. I/O Function Selection for Devices With Two Selection Bits – PxSEL0 and PxSEL1

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

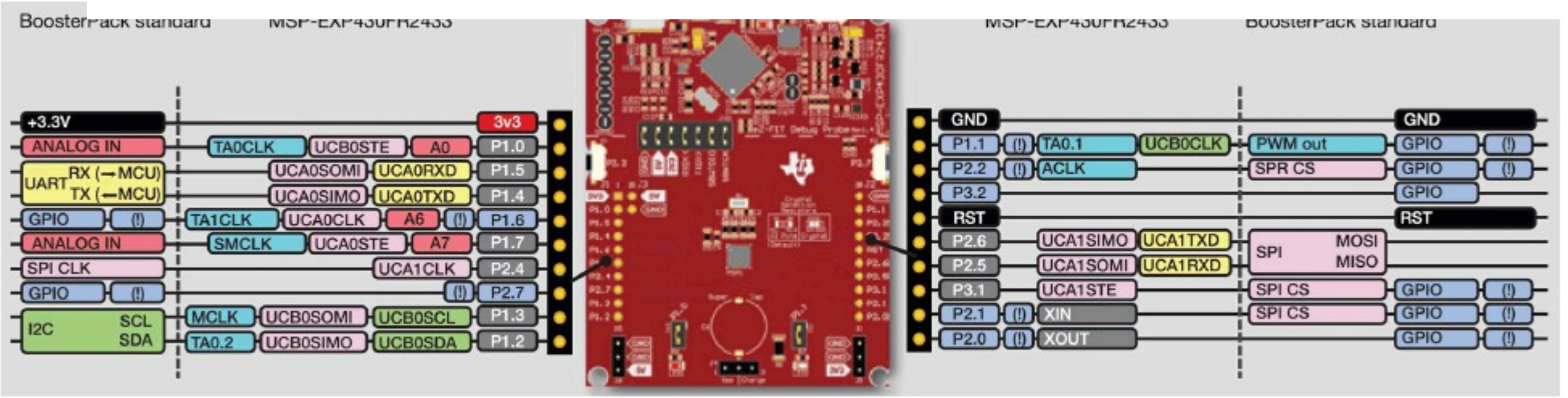


Figure 11. LaunchPad Kit to BoosterPack Module Connector Pinout

PxDir
  0 PxIn
  1 PxOut
PxREN (input mode)
  PxOUT 0 – pull down
          1 – pull up
PxSel
  00→10
ADCPCtLx – ADC Input

PxIE – Interrupt Enable Pin
PxIES – Rising or falling edge
PxIFG – Interrupt Flag (status)

GIE – General Interrupt Enable

Table 6-17. Port P1 (P1.0 to P1.7) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS[1]			
			P1DIR.x	P1SELx	ADCPCTLx[2]	JTAG
P1.0/UCB0STE/ TA0CLK/A0	0	P1.0 (I/O)	I: 0; O: 1	00	0	N/A
		UCB0STE	X	01	0	N/A
		TA0CLK	0	10	0	N/A
		A0/Veref+	X	X	1 (x = 0)	N/A
P1.1/UCB0CLK/TA0.1/ A1	1	P1.1 (I/O)	I: 0; O: 1	00	0	N/A
		UCB0CLK	X	01	0	N/A
		TA0.CCI1A	0	10	0	N/A
		TA0.1	1			
		A1	X	X	1 (x = 1)	N/A
P1.2/UCB0SIMO/ UCB0SDA/TA0.2/A2	2	P1.2 (I/O)	I: 0; O: 1	00	0	N/A
		UCB0SIMO/UCB0SDA	X	01	0	N/A
		TA0.CCI2A	0	10	0	N/A
		TA0.2	1			
		A2/Veref-	X	X	1 (x = 2)	N/A

PxDir
  0 PxIn
  1 PxOut
PxREN (input mode)
  PxOUT 0 – pull down
  1 – pull up
PxSel
  00→10
ADCPCtLx – ADC Input

PxIE – Interrupt Enable Pin
PxIES – Rising or falling edge
PxIFG – Interrupt Flag (status)

GIE – General Interrupt Enable

Each PxIFG bit is the interrupt flag for its corresponding I/O pin, and the flag is set when the selected input signal edge occurs at the pin. All PxIFG interrupt flags request an interrupt when their corresponding PxIE bit and the GIE bit are set. Software can also set each PxIFG flag, providing a way to generate a software-initiated interrupt.

- Bit = 0: No interrupt is pending
- Bit = 1: An interrupt is pending

7.2.6.2 Interrupt Edge Select Registers (PxIES)

Each PxIES bit selects the interrupt edge for the corresponding I/O pin.

- Bit = 0: Respective PxIFG flag is set on a low-to-high transition
- Bit = 1: Respective PxIFG flag is set on a high-to-low transition

7.2.6.3 Interrupt Enable Registers (PxIE)

Each PxIE bit enables the associated PxIFG interrupt flag.

- Bit = 0: The interrupt is disabled
- Bit = 1: The interrupt is enabled

6.13.2 Peripheral File Map

Table 6-24 lists the available peripherals and the register base address for each. Table 6-25 to Table 6-44 list the registers and address offsets for each peripheral.

Table 6-24. Peripherals Summary

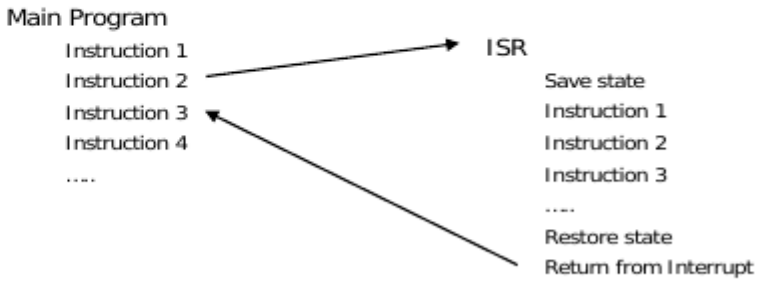
MODULE NAME	BASE ADDRESS	SIZE
Special Functions (See Table 6-25)	0100h	0010h
PMM (See Table 6-26)	0120h	0020h
SYS (See Table 6-27)	0140h	0040h
CS (See Table 6-28)	0180h	0020h
FRAM (See Table 6-29)	01A0h	0010h
CRC (See Table 6-30)	01C0h	0008h
WDT (See Table 6-31)	01CCh	0002h
Port P1, P2 (See Table 6-32)	0200h	0020h
Port P3 (See Table 6-33)	0220h	0020h
RTC (See Table 6-34)	0300h	0010h
Timer0_A3 (See Table 6-35)	0380h	0030h
Timer1_A3 (See Table 6-36)	03C0h	0030h
Timer2_A2 (See Table 6-37)	0400h	0030h
Timer3_A2 (See Table 6-38)	0440h	0030h
MPY32 (See Table 6-39)	04C0h	0030h
eUSCI_A0 (See Table 6-40)	0500h	0020h
eUSCI_A1 (See Table 6-41)	0520h	0020h
eUSCI_B0 (See Table 6-42)	0540h	0030h
Backup Memory (See Table 6-43)	0660h	0020h
ADC (See Table 6-44)	0700h	0040h

Interrupts Review

Code for ISR push button to toggle LED

What is an Interrupt?

- Reaction to something in I/O (human, comm link)
- Usually asynchronous to processor activities
- "interrupt handler" or "interrupt service routine" (ISR) invoked to take care of condition causing interrupt
  - Change value of internal variable (count)
  - Read a data value (sensor, receive)
  - Write a data value (actuator, send)



Interrupt review MSP430-Interrupts-posted.pdf

Interrupts

- Interrupts *preempt* normal code execution
  - Interrupt code runs in the *foreground*
  - Normal (e.g. **`main()`**) code runs in the *background*
- Interrupts can be *enabled* and *disabled*
  - *Globally*
  - *Individually* on a per-peripheral basis
  - *Non-Maskable* Interrupt (NMI)
- The occurrence of each interrupt is *unpredictable*
  - *When* an interrupt occurs
  - *Where* an interrupt occurs
- Interrupts are associated with a variety of on-chip and off-chip peripherals.
  - Timers, Watchdog, D/A, Accelerometer
  - NMI, change-on-pin (Switch)

Push a button, create an interrupt Switch 1 → P2.3

Interrupts

- Interrupts commonly used for
  - Urgent tasks w/higher priority than main code
  - Infrequent tasks to save polling overhead
  - Waking the CPU from sleep
  - Call to an operating system (software interrupt).
- Event-driven programming
  - The flow of the program is determined by events—i.e., sensor outputs or user actions (mouse clicks, key presses) or messages from other programs or threads.
  - The application has a main loop with event detection and event handlers.

Interrupt Flags

- Each interrupt has a flag that is raised (set) when the interrupt occurs.
- Each interrupt flag has a corresponding enable bit – setting this bit allows a hardware module to request an interrupt.
- Most interrupts are ***maskable***, which means they can only interrupt if

  Maskable – turn on/off with a flag

  1) enabled and
  2) the general interrupt enable (GIE) bit is set in the status register (SR).

Interrupt Vectors

- The CPU must know where to fetch the next instruction following an interrupt.
- The address of an ISR is defined in an *interrupt vector*.
- The MSP430 uses *vectored interrupts* where each ISR has its own vector stored in a *vector table* located at the end of program memory.
- Note: The *vector table* is at a fixed location (defined by the processor data sheet), but the ISRs can be located anywhere in memory.

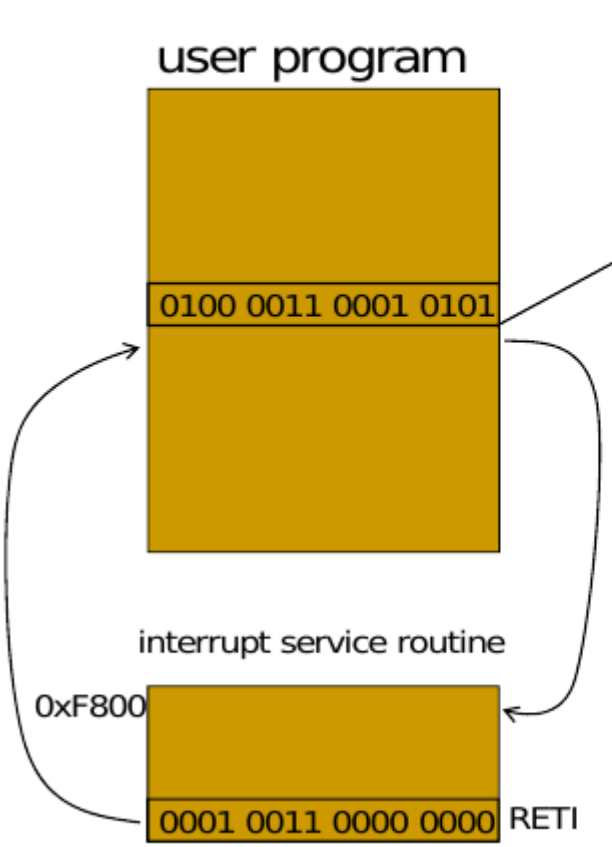
6.4 Interrupt Vector Addresses

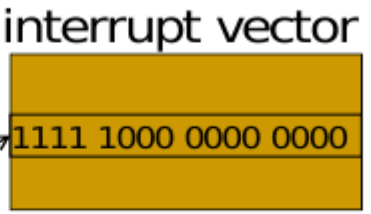
The interrupt vectors and the power-up start address are in the address range 0FFFFh to 0FF80h (see Table 6-2). The vector contains the 16-bit address of the appropriate interrupt-handler instruction sequence.

MSP430 Memory

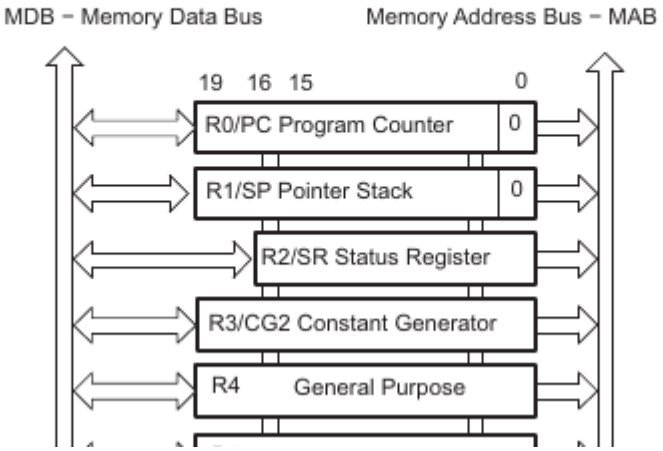
- Unified 64KB continuous memory map
- Same instructions for data and peripherals
- Program and data in Flash or RAM with no restrictions

Serving Interrupt Request





1. Lookup interrupt vector for ISR starting address.
2. Store information (PC and SR on Stack)
3. Transfer to service routine.
4. Restore information
5. Return (**`RETI: get old PC from stack`**).



CPU Registers

SLAU445G–October 2014–Revised August 2016
Submit Documentation Feedback

Memory	Size	Address	Description	Access
Flash	32KB	0xFFFF 0xFFC0	Interrupt Vector Table	Word
		0xFFBF 0x8000	Program Code	Word/Byte
SRAM	1KB	0x05FF 0x0200	Stack	Word/Byte
	256	0x01FF 0x0100	16-bit Peripherals Modules	Word
	240	0x00FF 0x0010	8-bit Peripherals Modules	Byte
	16	0x000F 0x0000	8-bit Special Function Registers	Byte

1.9.1.2 MSP430FR2433 Memory Map

Address Range	Name and Usage
00000h-00FFFh	Peripherals with gaps
00000h-000FFh	Reserved for system extension
00100h-00FEFh	Peripherals
00FF0h-00FF3h	Descriptor type(2)
00FF4h-00FF7h	Start address of descriptor structure
01800h-019FFh	Information Memory B
02000h-02FFFh	RAM 4KB SRAM
0C400h-0FFFFh	Program 15KB FRAM
0FF80h-0FFFFh	Interrupt Vectors

5.1 FRAM Introduction

FRAM is a nonvolatile memory that reads and writes like standard SRAM. The MSP430 FRAM features include:

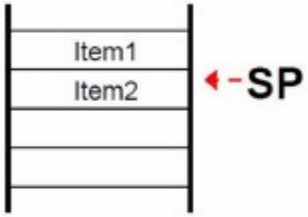
- Byte or word write access
- Automatic and programmable wait state control with independent wait state settings for access and cycle times
- Error correction code (ECC) with bit error correction, extended bit error detection, and flag indicators
- Cache for fast read
- Power control for disabling FRAM if it is not used

Figure 5-1 shows the block diagram of the FRAM Controller.

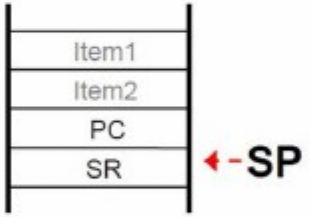
Processing an Interrupt…

1) Current instruction completed
2) MCLK started if CPU was off
3) Processor pushes program counter on stack
4) Processor pushes status register on stack
5) Interrupt w/highest priority is selected
6) Interrupt request flag cleared if single sourced
7) Status register is cleared
   - Disables further maskable interrupts (GIE cleared)
   - Terminates low-power mode
8) Processor fetches interrupt vector and stores it in the program counter
9) User ISR must do the rest!

Interrupt Stack

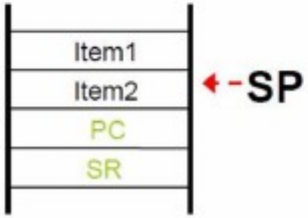


Prior to Interrupt Service Routine (=ISR)



ISR hardware - automatically

- Program Counter (= PC) pushed
- Status Register (= SR) pushed
- Interrupt vector moved to PC
- **GIE, CPUOFF, OSCOFF and SCG1 cleared**
- IFG flag cleared on single source flags



reti - automatically

- SR popped - *original*
- PC popped

Interrupt Service Routines

- Look superficially like a subroutine.
- However, unlike subroutines
  - ISR's can execute at unpredictable times.
  - Must carry out action and thoroughly clean up.
  - Must be concerned with shared variables.
  - Must return using ***reti*** rather than ***ret***.
- ISR must handle interrupt in such a way that the interrupted code can be resumed without error
  - Copies of all registers used in the ISR must be saved (preferably on the stack)

Interrupt Service Routines

- Well-written ISRs:
  - Should be *short* and *fast*
  - Should affect the rest of the system *as little as possible*
  - Require a *balance* between doing very little – thereby leaving the background code with lots of processing – and doing a lot and leaving the background code with nothing to do
- Applications that use interrupts should:
  - Disable interrupts *as little as possible*
  - *Respond to interrupts* as quickly as possible

Returning from ISR

- MSP430 requires 6 clock cycles before the ISR begins executing
  - The time between the interrupt request and the start of the ISR is called ***latency (plus time to complete the current instruction, 6 cycles, the worst case)***
- An ISR always finishes with the return from interrupt instruction (***reti***) requiring 5 cycles
  - The SR is popped from the stack
    - Re-enables maskable interrupts
    - Restores previous low-power mode of operation
  - The PC is popped from the stack
  - Note: if waking up the processor with an ISR, the new power mode must be set in the stack saved SR

Return From Interrupt

- Single operand instructions:

Mnemonic	Operation	Description
PUSH(.B or .W) src	SP-2→SP, src→@SP	Push byte/word source on stack
CALL dst	SP-2→SP, PC+2→@SP dst→PC	Subroutine call to destination
RETI	TOS→SR, SP+2→SP TOS→PC, SP+2→SP	Return from interrupt

- Emulated instructions:

Mnemonic	Operation	Emulation	Description
RET	@SP→PC SP+2→SP	MOV @SP+,PC	Return from subroutine
POP(.B or .W) dst	@SP→temp SP+2→SP temp→dst	MOV(.B or .W) @SP+,dst	Pop byte/word from stack to destination

P1 and P2 interrupts

- Only transitions (low to hi or hi to low) cause interrupts
- P1IFG & P2IFG (Port 1 & 2 Interrupt FlaG registers)
  - Bit 0: no interrupt pending
  - Bit 1: interrupt pending
- P1IES & P2IES (Port 1 & 2 Interrupt Edge Select reg)
  - Bit 0: PxIFG is set on low to high transition
  - Bit 1: PxIFG is set on high to low transition
- P1IE & P2IE (Port 1 & 2 Interrupt Enable reg)
  - Bit 0: interrupt disabled
  - Bit 1: interrupt enabled

Example P1 interrupt msp430x20x3_P1_02.c

```
#include  <msp430x20x3.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
  P1DIR |= 0x01;              // Set P1.0 to output direction
  P1IE |= 0x10;               // P1.4 interrupt enabled
  P1IES |= 0x10;              // P1.4 Hi/lo edge
  P1IFG &= ~0x10;             // P1.4 IFG cleared

  _BIS_SR(LPM4_bits + GIE);   // Enter LPM4 w/interrupt
}
// Port 1 interrupt service routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
  P1OUT ^= 0x01;                          // P1.0 = toggle
  P1IFG &= ~0x10;                         // P1.4 IFG cleared
}
```