M2V1

Blink and Button

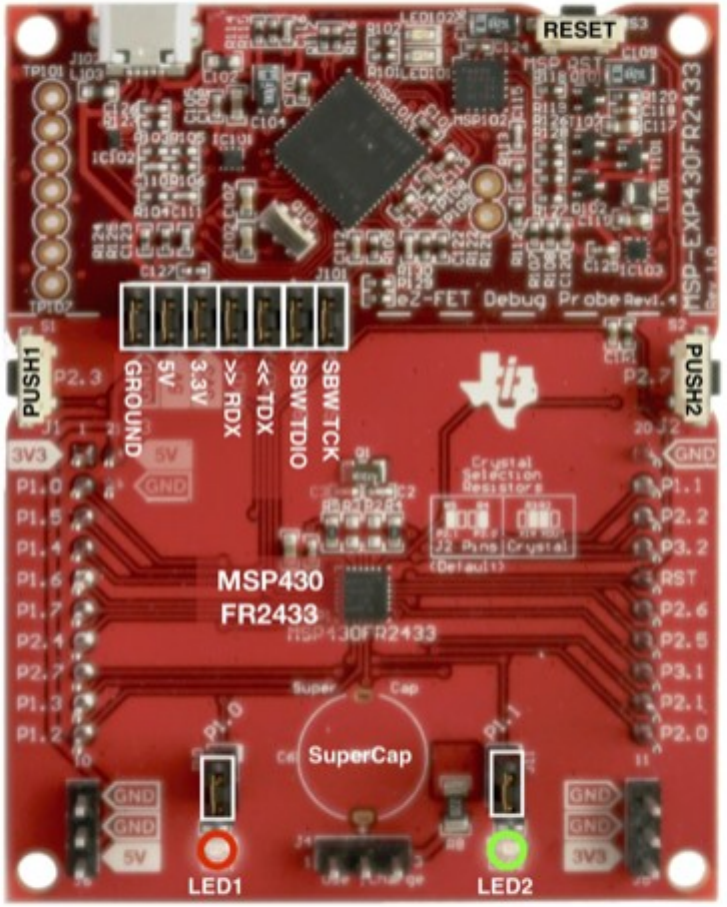
Learning through Code Examples

LaunchPad with MSP430FR2433

Revision 1.0

FRAM	16	KB
SRAM	4	KB

Serial	hardware
ADC	10 bits
Use pins numbers only!	
Default I²C = (0)	
Software I²C (1) master only	



+3.3V				1		+5V
LED1		A0	P1_0	2		GROUND
	RXD	A5	P1_5	3		
	TXD	A4	P1_4	4		
		A6	P1_6	5		
		A7	P1_7	6		
	SCK		P2_4	7		
PUSH2			P2_7	8		
	SCL (0)	A3	P1_3	9		
	SDA (0)	A2	P1_2	10		

GND
GND
+5V

Hardware
Pin number
Other Pins

I²C
Serial UART
SPI

analogRead()
digitalRead() and digitalWrite()
digitalRead(), digitalWrite() and analogWrite()

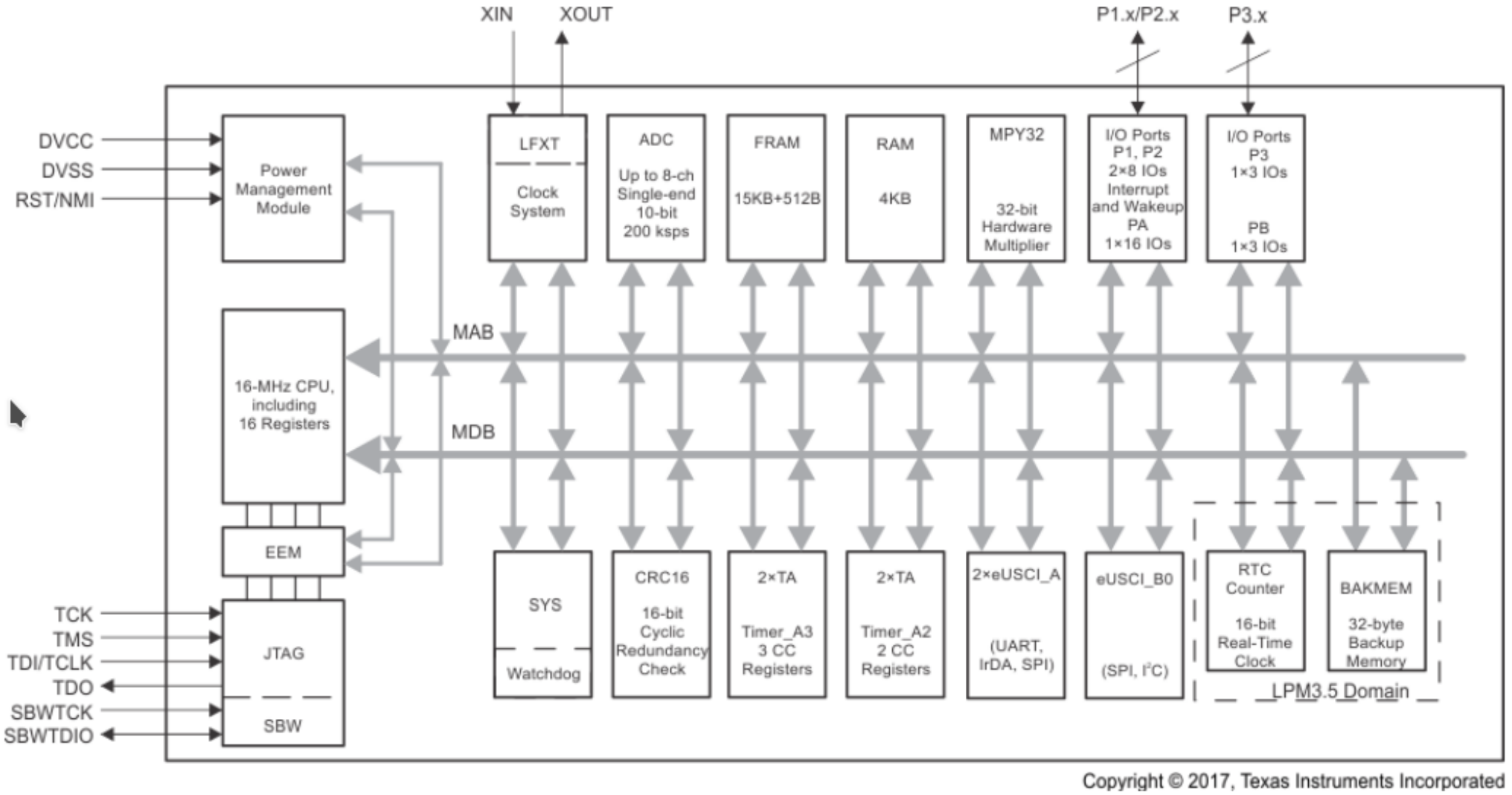
20				GROUND
19	P1_1	A1		LED2
18	P2_2			
17	P3_2			
16				RESET
15	P2_6	SDA (1)	MOSI	
14	P2_5	SCL (1)	MISO	
13	P3_1			
12	P2_1			
11	P2_0			

GND
GND
+3.3V

21	P2_3		PUSH1
		A12	TEMP

embeddedcomputing.weebly.com

version 0.9 2017-11-17 BETA



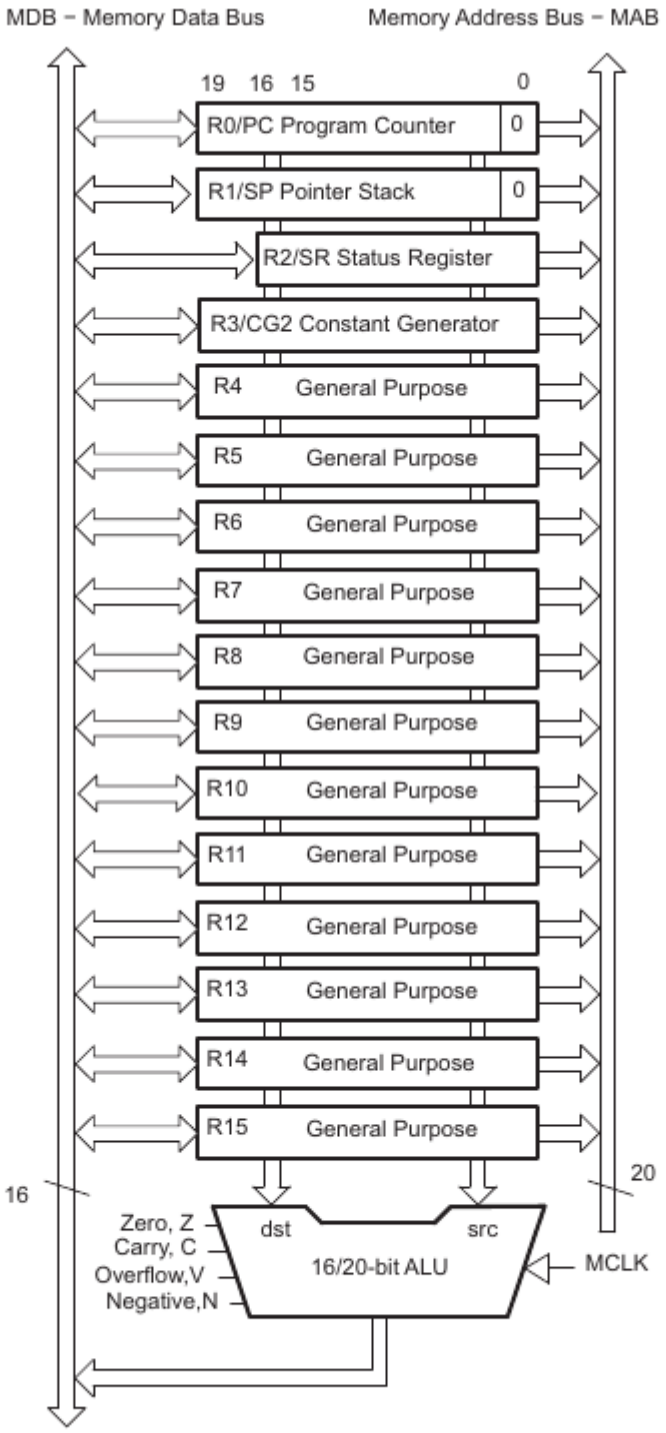


Figure 4-1. MSP430X CPU Block Diagram

4.2 Interrupts

The MSP430X has the following interrupt structure:

- Vectored interrupts with no polling necessary
- Interrupt vectors are located downward from address 0FFFEh.

The interrupt vectors contain 16-bit addresses that point into the lower 64KB memory. This means all interrupt handlers must start in the lower 64KB memory.

During an interrupt, the program counter (PC) and the status register (SR) are pushed onto the stack as shown in Figure 4-2. The MSP430X architecture stores the complete 20-bit PC value efficiently by appending the PC bits 19:16 to the stored SR value automatically on the stack. When the RETI instruction is executed, the full 20-bit PC is restored making return from interrupt to any address in the memory range possible.

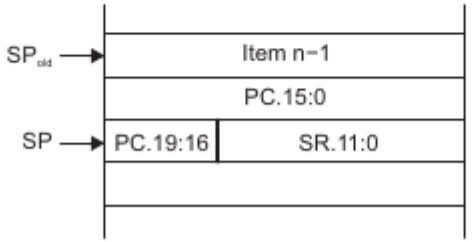


Figure 4-2. PC Storage on the Stack for Interrupts

4.3 CPU Registers

The CPU incorporates 16 registers (R0 through R15). Registers R0, R1, R2, and R3 have dedicated functions. Registers R4 through R15 are working registers for general use.

4.3.1 Program Counter (PC)

The 20-bit Program Counter (PC, also called R0) points to the next instruction to be executed. Each instruction uses an even number of bytes (2, 4, 6, or 8 bytes), and the PC is incremented accordingly. Instruction accesses are performed on word boundaries, and the PC is aligned to even addresses. Figure 4-3 shows the PC.

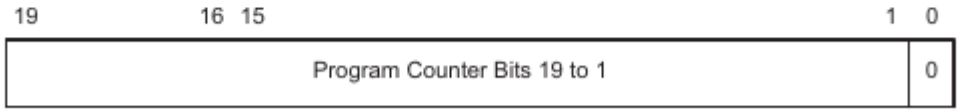


Figure 4-3. Program Counter

4.3.2 Stack Pointer (SP)

The 20-bit Stack Pointer (SP, also called R1) is used by the CPU to store the return addresses of subroutine calls and interrupts. It uses a predecrement, postincrement scheme. In addition, the SP can be used by software with all instructions and addressing modes. Figure 4-5 shows the SP. The SP is initialized into RAM by the user, and is always aligned to even addresses.

4.3.5 General-Purpose Registers (R4 to R15)

The 12 CPU registers (R4 to R15) contain 8-bit, 16-bit, or 20-bit values. Any byte-write to a CPU register clears bits 19:8. Any word-write to a register clears bits 19:16. The only exception is the SXT instruction. The SXT instruction extends the sign through the complete 20-bit register.

Figure 4-10 through Figure 4-14 show the handling of byte, word, and address-word data. Note the reset of the leading most significant bits (MSBs) if a register is the destination of a byte or word instruction.

Figure 4-10 shows byte handling (8-bit data, .B suffix). The handling is shown for a source register and a destination memory byte and for a source memory byte and a destination register.

4.4 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand use 16-bit or 20-bit addresses (see Table 4-3). The MSP430 and MSP430X instructions are usable throughout the entire 1MB memory range.

Table 4-3. Source and Destination Addressing

As, Ad	Addressing Mode	Syntax	Description
00, 0	Register	Rn	Register contents are operand.
01, 1	Indexed	X(Rn)	(Rn + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word.
01, 1	Symbolic	ADDR	(PC + X) points to the operand. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(PC) is used.
01, 1	Absolute	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word, or stored in combination of the preceding extension word and the next word. Indexed mode X(SR) is used.
10, –	Indirect Register	@Rn	Rn is used as a pointer to the operand.
11, –	Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards by 1 for .B instructions, by 2 for .W instructions, and by 4 for .A instructions.
11, –	Immediate	#N	N is stored in the next word, or stored in combination of the preceding extension word and the next word. Indirect autoincrement mode @PC+ is used.

4.6 Instruction Set Description

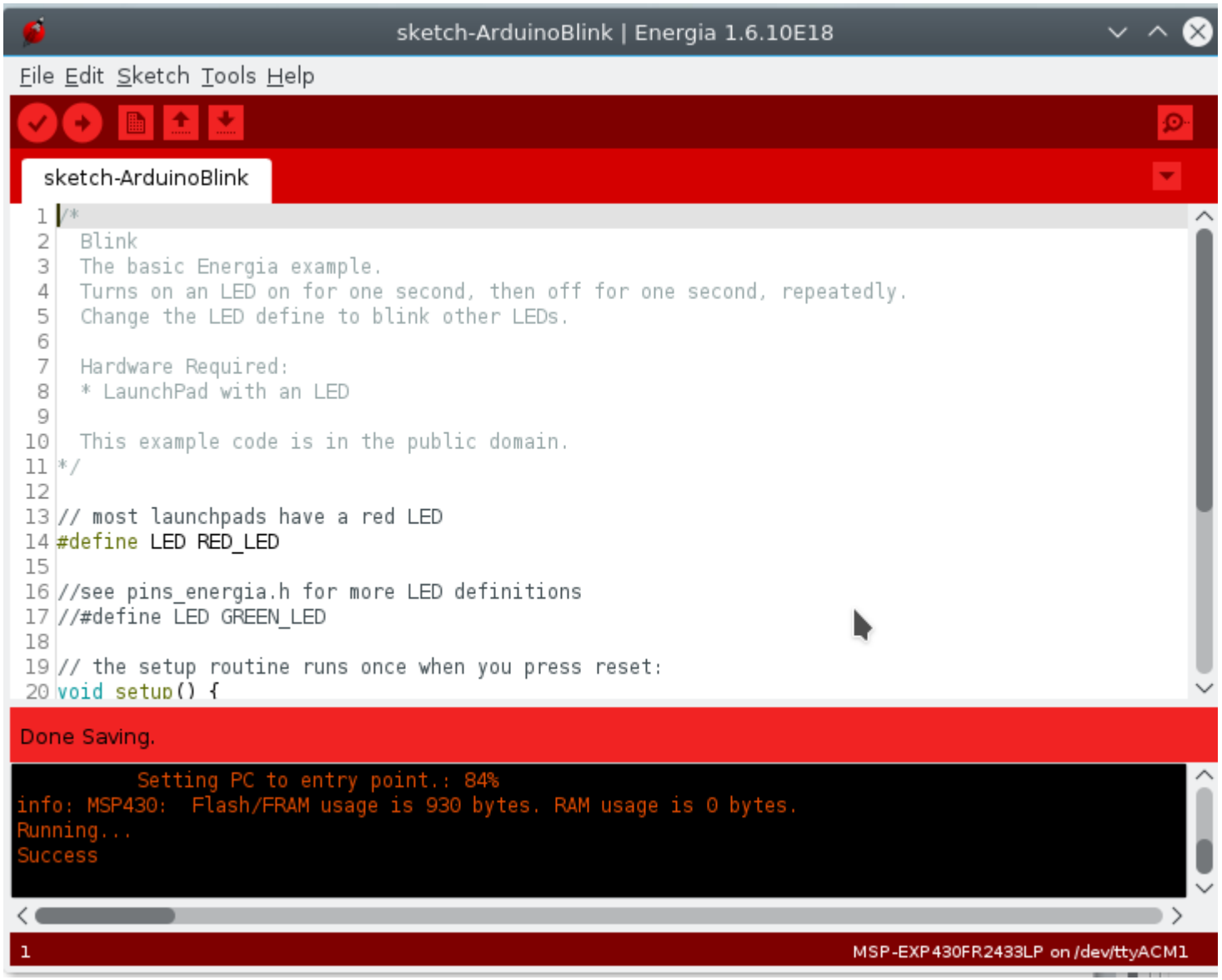
Table 4-20 shows all available instructions:

Table 4-20. Instruction Map of MSP430X

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA, CMPA, ADDA, SUBA, RRCM, RRAM, RLAM, RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A, POPM.A, PUSHM.W, POPM.W															
18xx 1Cxx	Extension word for Format I and Format II instructions															
20xx	JNE, JNZ															
24xx	JEQ, JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															


.know
I DON'T NEED TO KNOW EVERYTHING, I JUST NEED TO KNOW WHERE TO FIND IT, WHEN I NEED IT
- Albert Einstein -
MOSQUOTES.COM

- Use simple code examples
- Blinking LED
- Push Button, turn on LED
- Low Power Modes
- Interrupts
- Push button, ISR toggle LED


sketch-ArduinoBlink | Energia 1.6.10E18
File Edit Sketch Tools Help
sketch-ArduinoBlink
/*
Blink
The basic Energia example.
Turns on an LED on for one second, then off for one second, repeatedly.
Change the LED define to blink other LEDs.
Hardware Required:
* LaunchPad with an LED
This example code is in the public domain.
*/
// most launchpads have a red LED
#define LED RED_LED
//see pins_energia.h for more LED definitions
//#define LED GREEN_LED
// the setup routine runs once when you press reset:
void setup() {
Done Saving.
Setting PC to entry point.: 84%
info: MSP430: Flash/FRAM usage is 930 bytes. RAM usage is 0 bytes.
Running...
Success
1
MSP-EXP430FR2433LP on /dev/ttyACM1

sketch-ArduinoBlink.ino

```
/*
  Blink  - Arduino Default Version
  The basic Energia example.
  Turns on an LED on for one second, then off for one second, repeatedly.
  Change the LED define to blink other LEDs.

  Hardware Required:
  * LaunchPad with an LED

  This example code is in the public domain.
*/

// most launchpads have a red LED
#define LED RED_LED

//see pins_energia.h for more LED definitions
//#define LED GREEN_LED

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(LED, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}

```


sketch_CodeBlink | Energia 1.6.10E18
File Edit Sketch Tools Help
sketch_CodeBlink
volatile unsigned int i = 0;
// Initialize variables. This will keep count of how many
// cycles between LED toggles
int main(void)
{
//Setup
// This line of code turns off the watchdog timer,
// which can reset the device after a certain period of time.
WDTCTL = WDTPW + WDTHOLD;
// P1DIR is a register that configures the direction (DIR)
// of a port pin as an output or an input.
// After Power up all pins are input by default
// Set LSB P1.0 (RED LED) as output
P1DIR |= 0x01;
//Loop
Done uploading.
Setting PC to entry point.: 50%
info: MSP430: Flash/FRAM usage is 268 bytes. RAM usage is 0 bytes.
Running...
Success
22
MSP-EXP430FR2433LP on /dev/ttyACM1

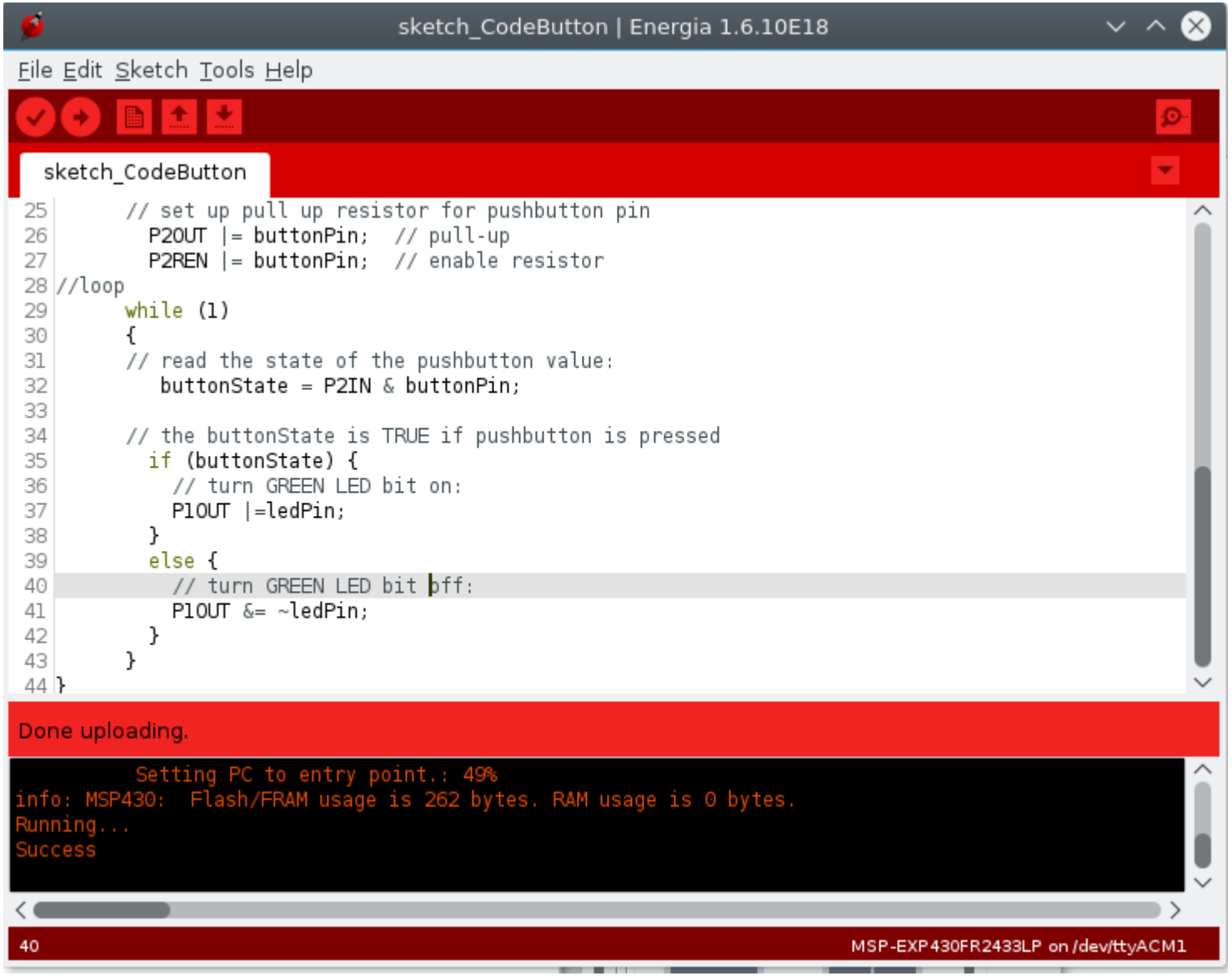
sketch_CodeBlink.ino

```
#include  <msp430.h>
  // Delay Loop Counter
  volatile unsigned int i = 0;
  #define outLED 0x01; // P1.0 (RED LED)

 int main(void)
 {
//Setup
      // This line of code turns off the watchdog timer,
      // which can reset the device after a certain period of time.
      WDTCTL = WDTPW + WDTHOLD;

      // P1DIR is a register that configures the direction (DIR)
      //  of a port pin as an output or an input.
      // After Power up all pins are input by default
      // Set LSB P1.0 (RED LED) as output ;
      P1DIR |= outLED;



//Loop
      while(1)
      {
       // turn the bit (RED_LED) on
         P1OUT |= outLED;

      // software delay loop
         for(i=0; i<64000; i++);

       // turn the bit (RED_LED) off
         P1OUT &= ~outLED;

     // software delay loop
         for(i=0; i<64000; i++);
      }

 }

```


sketch_ArduinoButton | Energia 1.6.10E18
File Edit Sketch Tools Help
sketch_ArduinoButton
30 // set pin numbers:
31 const int buttonPin = PUSH2;   // the number of the pushbutton pin
32 const int ledPin = GREEN_LED;   // the number of the LED pin
33
34 // variables will change:
35 int buttonState = 0;   // variable for reading the pushbutton status
36
37 void setup() {
38   // initialize the LED pin as an output:
39   pinMode(ledPin, OUTPUT);
40   // initialize the pushbutton pin as an input:
41   pinMode(buttonPin, INPUT_PULLUP);
42 }
43
44 void loop(){
45   // read the state of the pushbutton value:
46   buttonState = digitalRead(buttonPin);
47
48   // check if the pushbutton is pressed.
49   // if it is, the buttonState is HIGH:
Done Saving.
Setting PC to entry point.: 83%
info: MSP430: Flash/FRAM usage is 904 bytes. RAM usage is 0 bytes.
Running...
Success
3
MSP-EXP430FR2433LP on /dev/ttyACM1

sketch_ArduinoButton.ino

```
/*
  Button

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2.

 */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = PUSH2;     // the number of the pushbutton pin
const int ledPin =  GREEN_LED;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```


sketch_CodeButton | Energia 1.6.10E18
File Edit Sketch Tools Help
sketch_CodeButton
25 // set up pull up resistor for pushbutton pin
26 P2OUT |= buttonPin; // pull-up
27 P2REN |= buttonPin; // enable resistor
28 //loop
29 while (1)
30 {
31 // read the state of the pushbutton value:
32 buttonState = P2IN & buttonPin;
33
34 // the buttonState is TRUE if pushbutton is pressed
35 if (buttonState) {
36 // turn GREEN LED bit on:
37 P1OUT |=ledPin;
38 }
39 else {
40 // turn GREEN LED bit off:
41 P1OUT &= ~ledPin;
42 }
43 }
44 }
Done uploading.
Setting PC to entry point.: 49%
info: MSP430: Flash/FRAM usage is 262 bytes. RAM usage is 0 bytes.
Running...
Success
40
MSP-EXP430FR2433LP on /dev/ttyACM1

sketch_CodeButton.ino

```
/*
  Button Code for MSP430FR2433
  Copyright Herman Watson
 Creative Commons License
 */
  #include  <msp430.h>
// set pin numbers for MSP430FR2433:
const byte buttonPin = 0x80; // P2.7 (Button 2)
const byte ledPin =  0x02;   // P1.1 (Green LED)
volatile unsigned char buttonState = 0;         // variable for reading the pushbutton status

int main(void)
{
 //setup
       WDTCTL = WDTPW + WDTHOLD;   // stop the watchdog
      // initialize the LED pin as an output:
       P1DIR |= ledPin;

       // pushbutton pin already input by default after Power Up
      // set up pull up resistor for pushbutton pin
        P2OUT |= buttonPin;  // pull-up
        P2REN |= buttonPin;  // enable resistor
//loop
      while (1)
      {
      // read the state of the pushbutton value:
         buttonState = P2IN & buttonPin;

      // the buttonState is TRUE if pushbutton is pressed
        if (buttonState) {
          // turn GREEN LED bit on:
          P1OUT |=ledPin;
        }
        else {
          // turn GREEN LED bit off:
          P1OUT &= ~ledPin;
        }
      }
}

```