MSP430F5438 - System Cheat Sheet

CPUX

Table of Contents

MSP430X

The MSP430X CPU features include:

- RISC architecture
- Orthogonal architecture
- Full register access including program counter (PC), status register (SR), and stack pointer (SP)
- Single-cycle register operations
- Large register file reduces fetches to memory.
- 20-bit address bus allows direct access and branching throughout the entire memory range without paging.
- 16-bit data bus allows direct manipulation of word-wide arguments.
- Constant generator provides the six most often used immediate values and reduces code size.
- Direct memory-to-memory transfers without intermediate register holding
- Byte, word, and 20-bit address-word addressing

```
INST SRC, DST
```

Addressing Modes

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 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 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 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.

GPIO Ports and Pin Mapping

Port 1 (P1.x)

Pin	Phys Pin#	Function
P1.0	17	GPIO / TA0CLK / ACLK
P1.1	18	GPIO / TA0.0
P1.2	19	GPIO / TA0.1
P1.3	20	GPIO / TA0.2
P1.4	21	GPIO / TA0.3
P1.5	—	Not bonded
P1.6	—	Not bonded

Pin	Phys Pin#	Function
P1.7	—	Not bonded

Port 2 (P2.x)

Pin	Phys Pin#	Function
P2.0	22	GPIO / TA1.0
P2.1	23	GPIO / TA1.1
P2.2	24	GPIO / TA1.2
P2.3	25	GPIO / TA1.3
P2.4	26	GPIO / TA1.4
P2.5	27	GPIO / TA2.0
P2.6	28	GPIO / TA2.1
P2.7	29	GPIO / TA2.2

Port 3 (P3.x) – UART / SPI

Pin	Phys	Function
P3.0	30	UCA0TXD / UCA0SIMO
P3.1	31	UCA0RXD / UCA0SOMI
P3.2	32	UCA0CLK
P3.3	33	UCA0STE
P3.4	34	UCB0STE
P3.5	35	UCB0CLK
P3.6	36	UCB0SIMO / SDA
P3.7	37	UCB0SOMI / SCL

Port 4 (P4.x) – Timer B

Pin	Phys	Function
P4.0	38	TB0.0
P4.1	39	TB0.1
P4.2	40	TB0.2
P4.3	41	TB0.3
P4.4	42	TB0.4

Pin	Phys	Function
P4.5	43	TB0.5
P4.6	44	TB0.6
P4.7	45	GPIO

Port 5 (P5.x) – General I/O

Pin	Phys	Function
P5.0–P5.7	46–53	GPIO

Port 6 (P6.x) – ADC Inputs

Pin	Phys	Function
P6.0	54	ADC12 A0
P6.1	55	ADC12 A1
…	…	… up to P6.7 / A7

Port 7 (P7.x) – ADC Inputs A8–A15

| P7.0–P7.7 | Pins 62–69 | ADC12 A8–A15 |

Port 8,9,10 – General I/O, some SPI/UART

Port	Pins	Notes
P8	70–77	GPIO only
P9	78–85	Includes UCB2 I2C / SPI
P10	86–93	GPIO

Port J (PJ.x) – JTAG / Oscillator

PJ.x	Phys	Function
PJ.0	94	TDO / SBWTCK
PJ.1	95	TDI / TCLK
PJ.2	96	TMS
PJ.3	97	TCK
PJ.4	98	LFXIN
PJ.5	99	LFXOUT
PJ.6	100	XT2IN

PJ.x	Phys	Function
PJ.7	1	XT2OUT

GPIO Registers Overview

Register	Purpose
`PxDIR`	0 = input, 1 = output
`PxOUT`	Output state OR pull-up selection if REN = 1
`PxIN`	Read live pin logic level
`PxREN`	Enable pull-up/down resistor
`PxSEL / PxSEL2`	Select peripheral vs GPIO
`PxIE`	Interrupt enable
`PxIFG`	Interrupt flag
`PxIES`	Interrupt edge (0 = rising, 1 = falling)

Example: input with pull-up on P1.3

```
P1DIR &= ~BIT3;
P1REN |= BIT3;
P1OUT |= BIT3; // pull-up enabled
```

MSP430 Calling Convention

- First 4 parameters: `R12–R15`
- Return value in: `R12`
- Frame Pointer: `R11` if used

Reg	Name	Purpose
R0	PC	Program Counter
R1	SP	Stack Pointer
R2	SR	Status Register
R3	CG2	Constant generator
R4–R11	—	Callee-saved registers
R12–R15	—	Function args + return values

Stack Frame Layouts

Function Call:

```
Higher Addresses
+---------------------+
| Local variables     |
| Saved R4–R11        |
| Return Address (PC)|
| Saved SR            |
+---------------------+
Lower Addresses (SP)
```

Interrupt Entry:

```
SP−2 → Saved SR
SP−4 → Saved PC
```

Interrupt System

- Hardware automatically pushes SR and PC.
- ISR ends with `RETI` (pops PC + SR).
- Example:

```
#pragma vector=PORT1_VECTOR
__interrupt void Port1_ISR(void) {
    P1IFG &= ~BIT3;
}
```

Clock Pins

Signal	Pin	Port
LFXIN	Pin 98	PJ.4
LFXOUT	Pin 99	PJ.5
XT2IN	Pin 100	PJ.6
XT2OUT	Pin 1	PJ.7
ACLK out	P1.0 / P11.0	
SMCLK out	P3.4 / P4.0 (depends on SEL config)	

Memory Map

Region	Address Range
Special Function Registers	0x0000–0x01FF
RAM	0x1C00–0x23FF
Info Flash	0x1800–0x19FF
Main Flash	0x4400–0xFFFF
Interrupt Vectors	0xFFC0–0xFFFF

Example C Code to Setup UART:

```
// Some basic UART Setup
#include <msp430.h>

void Port_Mapping(void);


int main(void)
{
  WDTCTL = WDTPW | WDTHOLD;                     // Stop WDT

  while(BAKCTL & LOCKBAK)                        // Unlock XT1 pins for
operation
     BAKCTL &= ~(LOCKBAK);
  UCSCTL6 &= ~(XT1OFF);                         // XT1 On
  UCSCTL6 |= XCAP_3;                            // Internal load cap
  // Loop until XT1 fault flag is cleared
  do
  {
    UCSCTL7 &= ~(XT2OFFG | XT1LFOFFG | DCOFFG);
                                                // Clear XT2,XT1,DCO fault
flags
    SFRIFG1 &= ~OFIFG;                          // Clear fault flags
  }while (SFRIFG1&OFIFG);                       // Test oscillator fault flag

  Port_Mapping();

  P2SEL |= 0x03;                                // Assign P2.0 to UCA0TXD
and...
  P2DIR |= 0x03;                                // P2.1 to UCA0RXD

  UCA0CTL1 |= UCSWRST;                          // **Put state machine in
reset**
  UCA0CTL1 |= UCSSEL_1;                         // CLK = ACLK
  UCA0BR0 = 0x03;                               // 32kHz/9600=3.41 (see User's
Guide)
  UCA0BR1 = 0x00;                               //
  UCA0MCTL = UCBRS_3|UCBRF_0;                   // Modulation UCBRSx=3,
UCBRFx=0
```

```
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state
machine**
  UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

  __bis_SR_register(LPM3_bits | GIE);       // Enter LPM3, interrupts
enabled
  __no_operation();                         // For debugger
}

// Echo back RXed character, confirm TX buffer is ready first
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
    while (!(UCA0IFG&UCTXIFG));             // USCI_A0 TX buffer ready?
    UCA0TXBUF = UCA0RXBUF;                  // TX -> RXed character
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}


void Port_Mapping(void)
{
  // Disable Interrupts before altering Port Mapping registers
  __disable_interrupt();
  // Enable Write-access to modify port mapping registers
  PMAPPWD = 0x02D52;

  #ifdef PORT_MAP_RECFG
  // Allow reconfiguration during runtime
  PMAPCTL = PMAPRECFG;
  #endif

  P2MAP0 = PM_UCA0TXD;
  P2MAP1 = PM_UCA0RXD;

  // Disable Write-Access to modify port mapping registers
  PMAPPWD = 0;
  #ifdef PORT_MAP_EINT
  __enable_interrupt();                     // Re-enable all interrupts
  #endif
}
```

Documentation

DevTI For MSP430F5438

DriverLib

Data Structs

Data Sheet