M12 - Peripheral Interrupts as Tasks and code Building Blocks

Peripheral Interrupt Tasks

MSP430FR2433
Sections reviewed:
Software Building Blocks

GPIO
Clock System
Timer
- Count
- Compare
- Capture
- PWM

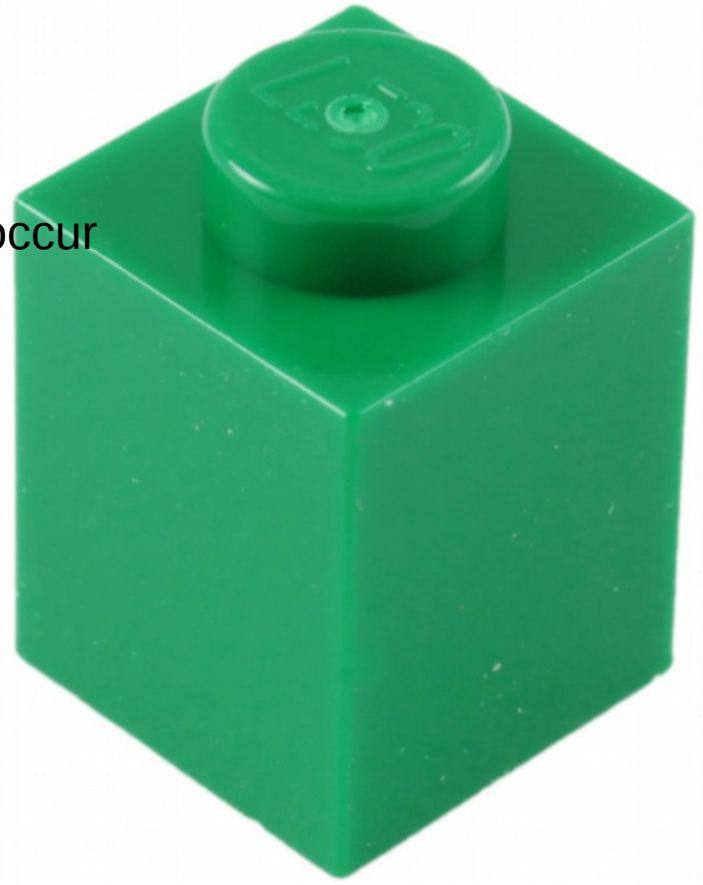
UART
ADC I & II

Interrupt Tasks – wake up from Low Power Mode

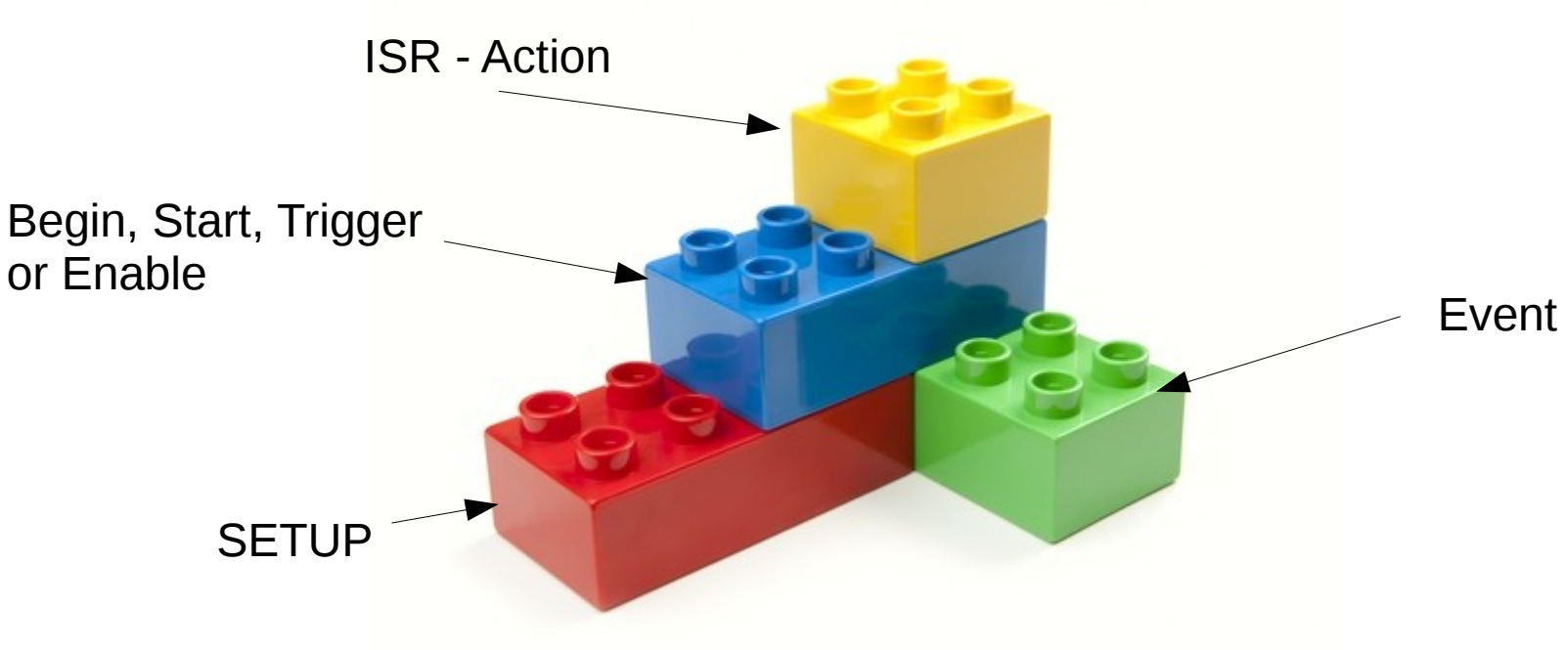
Each one of these Building Blocks is presented

as an interrupt TASK to be used by placing the CPU

into Low Power Standby and waiting for an interrupt to occur



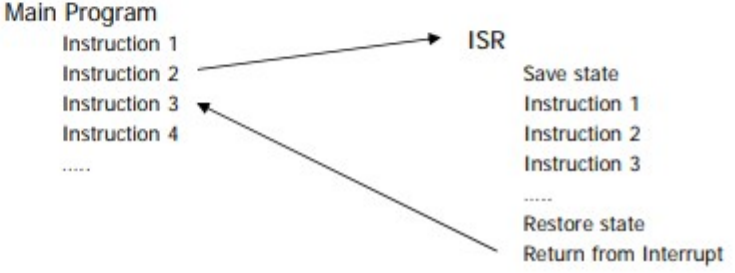
TASKS - each peripheral interrupt TASK has similar structure for coding



What is an Interrupt?

Interrupt Task Event

- 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)



https://courses.cs.washington.edu/courses/cse466/12au/calendar/03-Interrupts-posted.pdf

Interrupt Task Action

- 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 separate event detection and event handlers.

TASK SETUP – control register bit settings

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

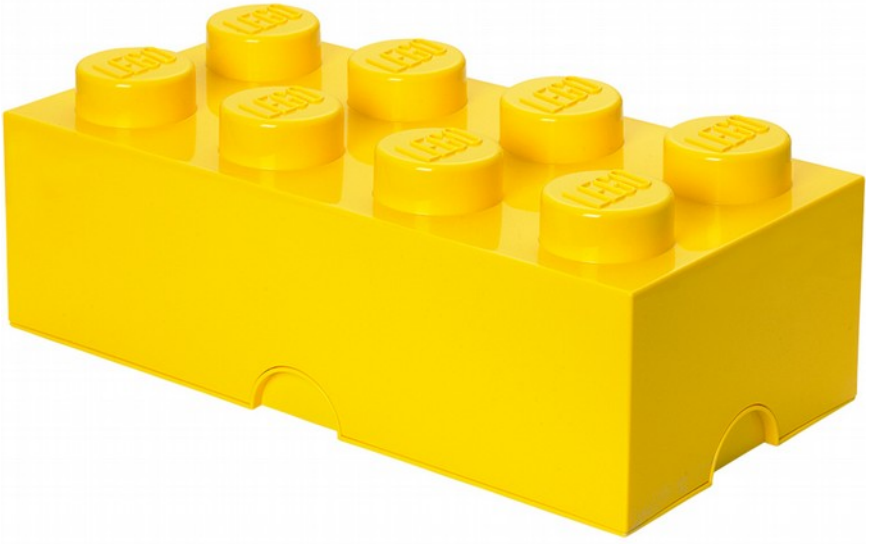
TASK SETUP - vector

- 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.

TASK ISR – code

Interrupt Service Routines

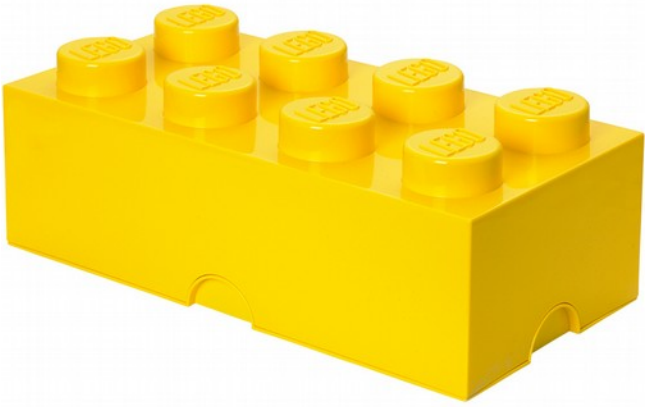
- Look superficially like a subroutine.
- However, unlike subroutines
  - ISR's can execute at unpredictable times.
  - Must carry out action and thoroughly clean up.
  - Must be concerned with shared variables.
  - Must return using ***reti*** rather than ***ret***.
- ISR must handle interrupt in such 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)



TASK ISR - ACTIONS

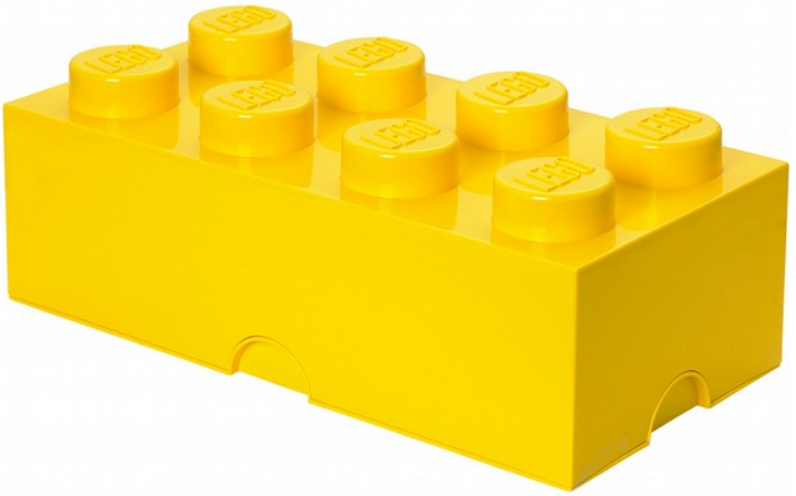
Interrupt Service Routines

- 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



TASK ISR - reti

- 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



TASK ENABLE

Begin
- Load Tx Buffer

Start
- Start timer

Trigger
- Push the button

Enable
- Turn on Peripheral IE
- Load the Mouse Trap

TASK EVENT

GPIO

- Push Button
- Mouse eats cheese

Timer

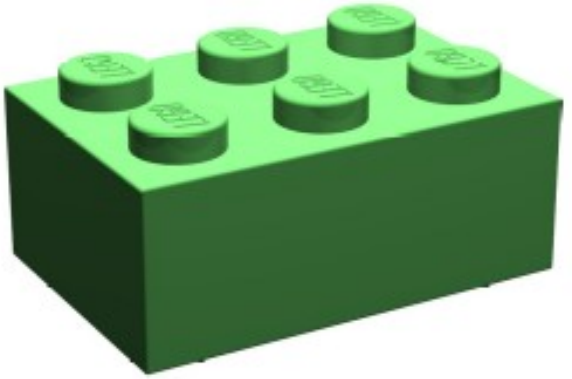
- TA0 Overflow
- CCRx Equal
- CCRx Capture

UART

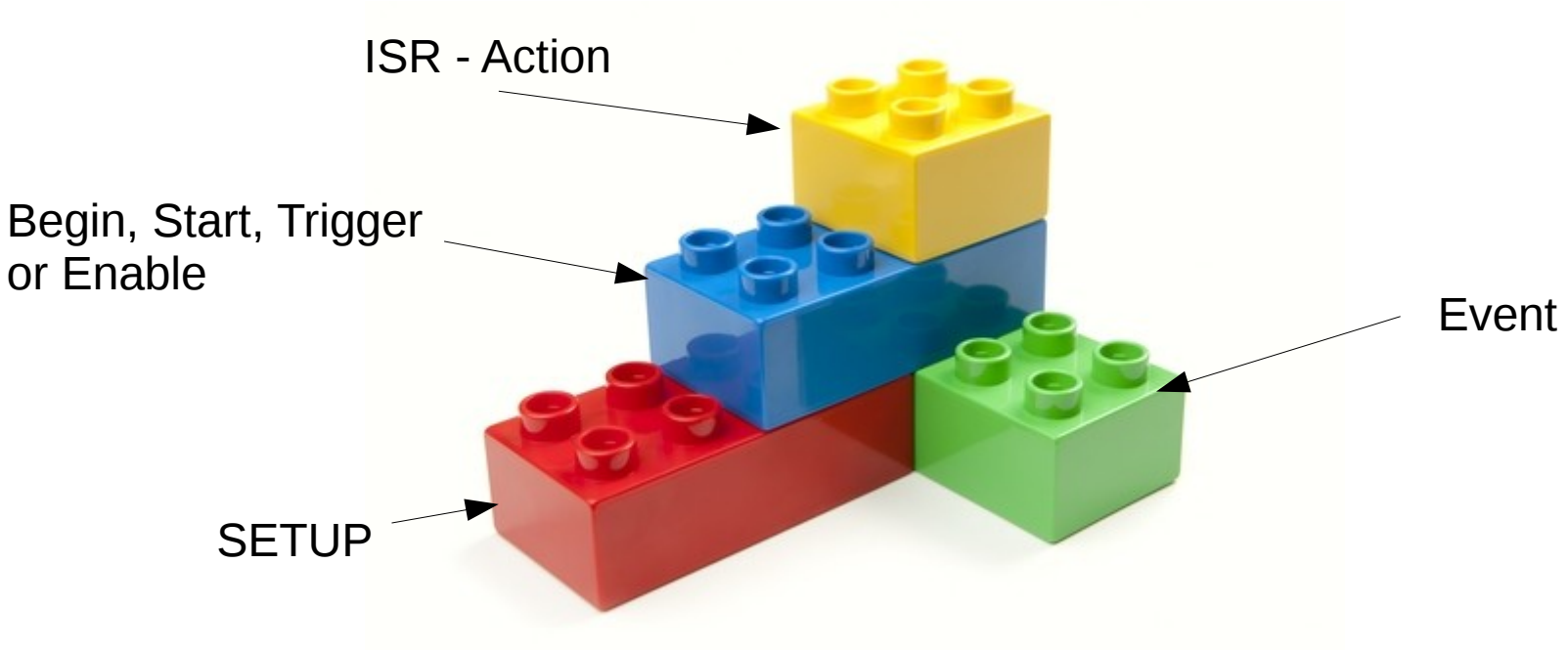
- TX Empty
- RX Full

ADC

- Conversion Done



TASKS - each peripheral interrupt TASK has similar structure for coding



Example TASK - M3-GPIO: Interrupt on Push the Button

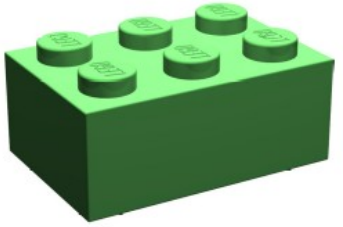
SETUP

```
46
47      // P2.3 input switch (all writes are 8 bits)
48      P2DIR &= ~BIT3;    // set P2.3 as input bit (input default power-up)
49      P2OUT |= BIT3;     // BIT3 on, set outbit to pull-up
50      P2REN |= BIT3;     // BIT3 on, Enable internal pull up register
51
52      P2IES |= BIT3;     // One is Falling edge
```

Enable

```
53
54      P2IE |= BIT3;      // BIT3 on, P2.3 interrupt enabled
55      P2IFG &= ~BIT3;    // BIT3 off, P2.3 IFG cleared
```

Event:
Push the Button



ISR

```
62
63  // Port 2 interrupt service routine
64  #pragma vector=PORT2_VECTOR
65  __interrupt void Port_2(void)
66  {
67      P1OUT ^= BIT0; // P1.0 = toggle (xor)
68      P2IFG &= ~BIT3; // P2.3 IFG off (cleared)
69  }
```

http://web.eng.fiu.edu/watsonh/IntroMicros/M3-GPIO/sketch_ButtonISR.ino

M11 – Clocked ADC Example: sketch_RT_ADC4Exam.ino

GPIO TASK: Configure RED LED on GPIO: No ISR

```
14      P1DIR |= BIT0; // RED LED  P1.0
```

SETUP:
Line 14

Clock System TASK: Configure – 1MHz MCLK, SMCLK, & ACLK 32768Hz: No ISR

```
19  //Clock System  ACLK = 32786, MCLK = SMCLK = 1MHz
20        __bis_SR_register(SCG0);  // disable FLL
21      CSCTL3 |= SELREF__REFOCLK; // Set ref source
22      CSCTL0 = 0; // clear DCO and MOD regs
23      CSCTL1 &= ~(DCORSEL_7); // Clear freq select
24      CSCTL1 |= DCORSEL_3;  // Set DCOCLK = 8MHz
25      CSCTL2 = FLLD_1 + 121; //Set div & Remainder
26      __delay_cycles(3);
27      __bic_SR_register(SCG0); // enable & lock FLL
28      while(CSCTL7 & (FLLUNLOCK0 | FLLUNLOCK1));
29      CSCTL4 = SELMS__DCOCLKDIV | SELA__XT1CLK;
30      CSCTL5 |= DIVM1; // DCODIV/2 = 1MHz
```

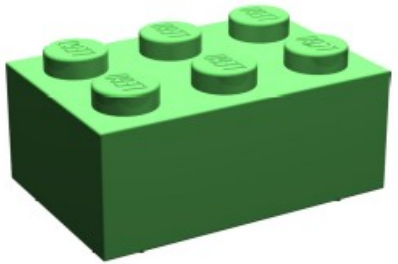
SETUP:
Lines 19-30

Timer0_A3 TASK: Interrupt 10 times per second

```
42  // Timer0_A3 Setup  CCR0 ISR 10/second:
43      TA0EX0 |= TAIDEX_3; // SMCLK/8/4 = 31250 Hz
44      TA0CCR0 = 3125;  // 10 per second
45
46      TA0CTL = TASSEL_2 | MC_1 | ID_3;//SMCLK/8, UP
47      TA0CCTL0 |= CCIE;  // TACCR0 int enabled
```

SETUP:
Lines 42-46

ENABLE:
Line 47



Event:
TA0 == CCR0

```
52  // Timer A0 interrupt service routine
53  #pragma vector = TIMER0_A0_VECTOR
54  __interrupt void Timer_A (void)
55  {
56      P1OUT ^= BIT0;
57      ADCCTL0 |= ADCENC | ADCSC; // conversion start
58  }
```

ISR: CCR0
Lines 54-58
Action:
Start ADC

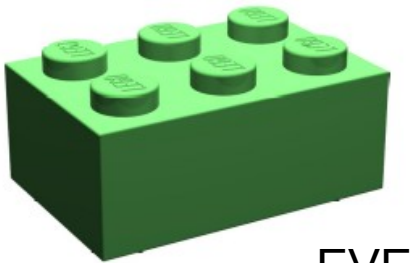
ADC TASK: Conversion 10/second (Triggered by Timer0_A3)

```
34  // Configure ADC10 A1 on P1.1
35    SYSCFG2 |= ADCPCTL1;
36    ADCCTL0 |= ADCSHT_2 | ADCON;// ADC ON, 16 clks
37    ADCCTL1 |= ADCSHP | ADCSSEL0; //CLK = MODOSC
38    ADCCTL2 |= ADCRES;  // 10-bit conversion
39    ADCMCTL0 |= ADCINCH_1; //A1 input; Vref=AVCC
40    ADCIE |= ADCIE0;  // Enable complete interrupt
```

SETUP:
Lines 35-39

ENABLE:
Line 40

```
52  // Timer A0 interrupt service routine
53  #pragma vector = TIMER0_A0_VECTOR
54  __interrupt void Timer_A (void)
55  {
56     P1OUT ^= BIT0;
57     ADCCTL0 |= ADCENC | ADCSC; // conversion start
58  }
```



EVENT:
Line 57

ISR: next slide

ADC TASK: Conversion 10/second (Triggered by Timer0_A3)

```
 83  #pragma vector=ADC_VECTOR
 84  __interrupt void ADC_ISR(void)
 85  {
 86    switch(ADCIV)
 87  {
 88    case ADCIV_NONE:
 89      break;
 90    case ADCIV_ADCOVIFG:
 91      break;
 92    case ADCIV_ADCTOVIFG:
 93      break;
 94    case ADCIV_ADCHIIFG:
 95      break;
 96    case ADCIV_ADCLOIFG:
 97      break;
 98    case ADCIV_ADCINIFG:
 99      break;
100    case ADCIV_ADCIFG:
101      ADC_Result = ADCMEM0;
102      sprintf(OutStr,"%d\n",ADC_Result);
103      UARTPutString(OutStr); // output string
104    break;
105   default:
106    break;
107      }
108  }
```

ISR: ADC Conversion complete
Lines 83-108

Action:
Create output string
Start string output

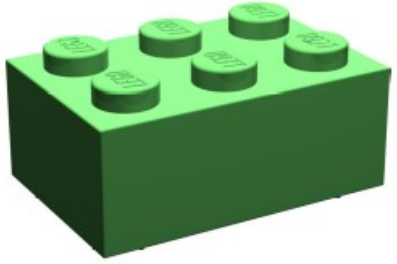
UART Task: Print output string of ADC Value (10/second)

```
110  void UARTSetup (void)
111  {
112    P1SEL0 |= BIT4 | BIT5; //UART pins function
113    // Configure UART
114    UCA0CTLW0 |= UCSWRST;   // reset UART
115    UCA0CTLW0 |= UCSSEL__SMCLK; // use SMCLK input
116      UCA0BR0 = 104;     // 1MHz SMCLK/9600 BAUD
117      UCA0MCTLW = 0x1100;// remainder of Baud Rate
118    UCA0CTLW0 &= ~UCSWRST;
119  }
```

SETUP:
Lines 110-119

```
121  void UARTPutString(const char* strptr)
122  { // load TxBuf with ptr to first char
123      TxPtr = strptr;
124      UCA0IFG |=UCTXIFG; // set empty IFG
125      UCA0IE |= UCTXIE;  // int enable, ISR NOW
126  }
```

ENABLE:
Lines 121-126
Output First Char



EVENT:
New string created fromADC value

ISR next slide

UART Task: Print output string of ADC Value (10/second)

```
60  #pragma vector=USCI_A0_VECTOR
61  __interrupt void USCI_A0_ISR(void)
62  {
63    switch(UCA0IV)
64    {
65     case USCI_NONE: break;
66     case USCI_UART_UCRXIFG:
67      while(!(UCA0IFG&UCTXIFG));
68      UCA0TXBUF = UCA0RXBUF;
69       __no_operation();
70     break;
71     case USCI_UART_UCTXIFG:
72       if(!(*TxPtr))  // if EOS, stop
73         UCA0IE &= ~UCTXIE;  // turn off interrupt
74       else
75         UCA0TXBUF = *TxPtr++ ;
76      break;
77     case USCI_UART_UCSTTIFG: break;
78     case USCI_UART_UCTXCPTIFG: break;
79     default: break;
80    }
81  }
```

ISR:
TXBUF is empty
Lines 60-81

Action:
Load next char
or
Stop if EOS (end-of-string)


N25ILF

Module 11 – Clocked ADC with output stream –
Whole picture – Helicopter View

SetUp:
    Init CLK, Timer0_A3, ADC, UART

Loop:
    Enable Interrupts, Low Power Mode (Standby)

Task: 1

Timer0_A0 ISR: TA0==CCR0 (10/sec)
    Start ADC sample on ADC CH1

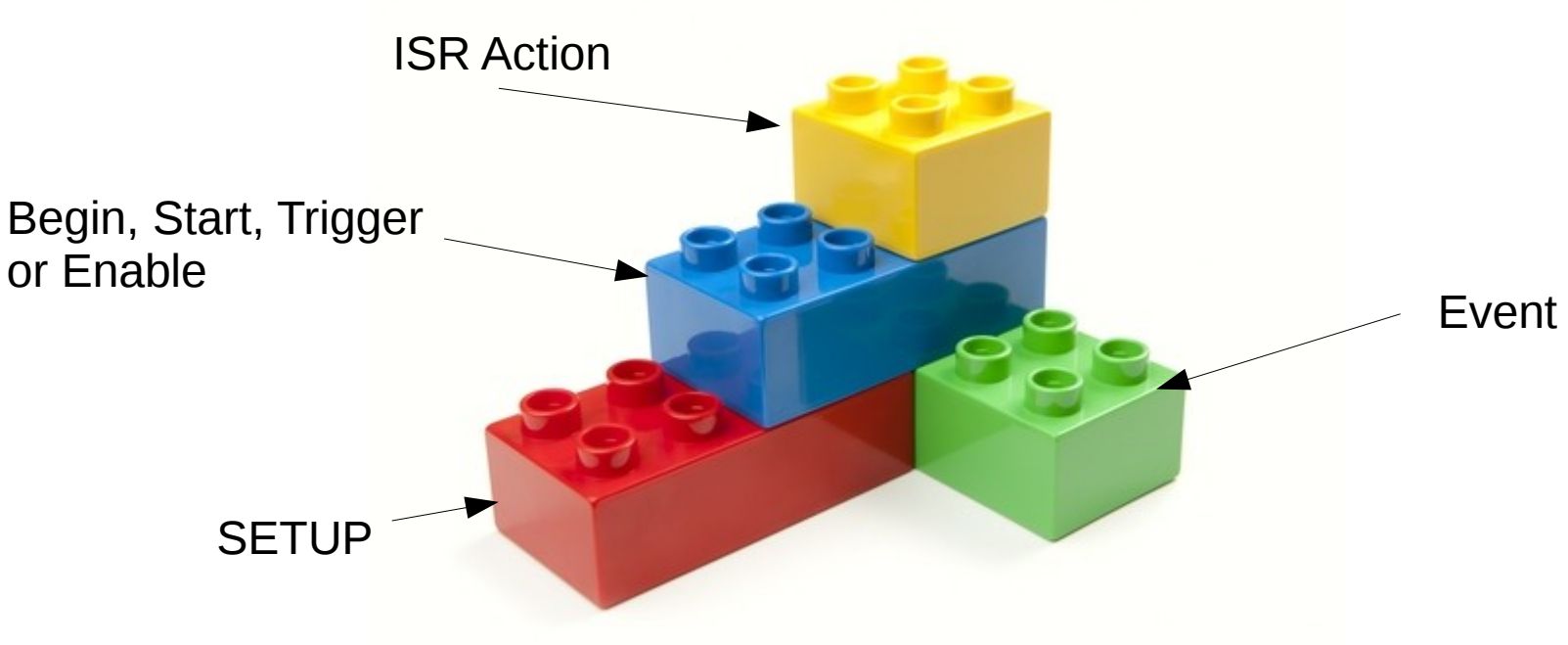
Task: 2

ADC ISR: Conversion completed
    Create Output String of ADC Value
    Start TX-IE

Task: 3

TX ISR: TXBUF empty
    Get next char
    End-Of-String?
        No – RETI
        Yes- Turn off TX-IE

Tasks - each interrupt has common building blocks structure for coding



New Perspective: Develop a view of code sections as building blocks