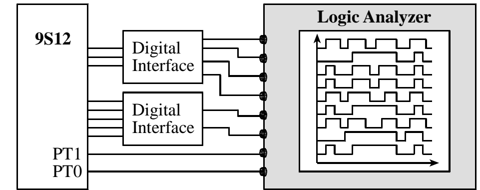
Lecture 4 objectives

- debugging techniques: dumps, monitor
- draw pictures showing elements on the stack
- debugging from an assembly language perspective

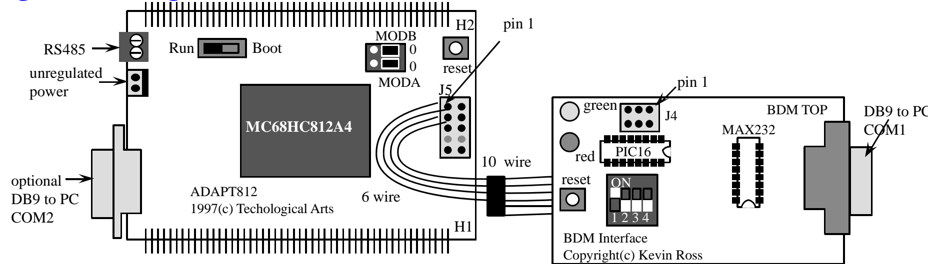


Hardware debugging tools

Logic analyzer

Multiple channel, digital, storage scope
Flexible method of triggering

Background Debug Module (BDM)



TI's Debug Module (Spy Bi-ware)

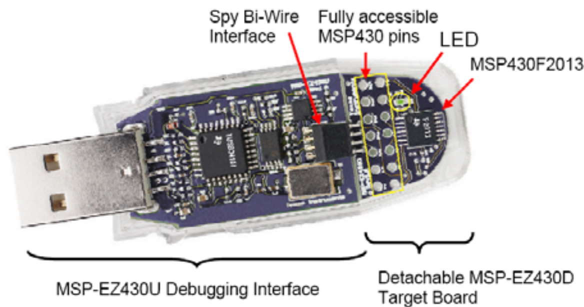


Figure 1. eZ430-F2013 Development Tool

Industry Standard Debug Module (JTAG)

Debugging from an assembly perspective

```
void fun(void){
short n;  // where is n?
  n = 1;
  PTT = n;
  PTP = 2;

void main(void){
  fun();
}

  ldaa #1
  staa PTT
  inca
  staa PTP
```

Using a hardware debugger (BDM) while running

- observe memory/ports
- change memory/ports values
- program EEPROM
- start execution
- reset computer

When stopped

- observe registers
- change register values

Using a software debugger (serial monitor) while running

Serial transmission from PC to 9S12 causes a SCI0 interrupt
Your program halts
For every item, 5+n transmissions to populate
Your program resumes

Serial monitor adds a 19 cycle overhead on all interrupts

A debugging instrument

software code that is added to the program
the purpose of debugging
e.g., print statement
using editor/assembler/loader, one adds instrument

Leave the instruments

Because this is the way it was proven to work
May need more testing

Choose one of the following techniques

- Place all instruments in the first column, they are easy to see
- Define instruments with specific pattern in their names
- Use instruments that test a run time global flag
  - leaves a permanent copy of the debugging code
  - causing it to suffer a runtime overhead
  - simplifies “on-site” customer support.
- Use conditional compilation (or conditional assembly)
  - Easy to remove all instruments

```
#define DEBUG 1     // insert/skip instruments
#ifndef bit
#define bit(x)     (1 << (x))
#endif
#if DEBUG
#define SETBIT(port,b)  (port|=bit(b))
#define CLRBIT(port,b)  (port&=~bit(b))
#define FLIPBIT(port,b) (port^=bit(b))
#else
#define SETBIT(port,b)
#define CLRBIT(port,b)
#define FLIPBIT(port,b)
#endif
```

Add a LED monitor

Question: Is my program running?
Method: Add instruments, assemble, download, run
Initialization

```
   DDRP |= 0x80;
```

Debugging Instrument

```
   FLIPBIT(PTP,7);
   PTP ^= 0x80;
```

Results: No it is not running?
Action: Fix the bug, assemble, download, run
Results: It is running, but still has a bug.

Run OC project, see LED monitor

```
unsigned short static volatile Count0;
interrupt 8 void TOC0handler(void){
// executes at 100 Hz
  TFLG1 = 0x01;         // acknowledge OC0
  Count0++;
  TC0 = TC0+10000;      // 10 ms
  PTT ^= 0x01;          // debugging monitor
}
//----------------OC_Init0-----------------
// arm output compare 0 for 100Hz interrupt
// Input: none
// Output: none
void OC_Init0(){
  Count0 = 0;     // debugging monitor
  DDRT |= 0x01;   // debugging monitor
  TIOS |= 0x01;   // TC0 is output compare
  TIE  |= 0x01;   // arm OC0
  TSCR1 = 0x80;   // Enable TCNT
  TSCR2 = 0x03;   // divide by 8 prescale
  PACTL = 0;      // timer prescale
/* Bottom three bits of TSCR2 (PR2,PR1,PR0) set TCNT period
    divide  FastMode(24MHz)    Slow Mode (8MHz)
000   1     42ns  TOF  2.73ms  125ns TOF  8.192ms
001   2     84ns  TOF  5.46ms  250ns TOF 16.384ms
010   4    167ns  TOF  10.9ms  500ns TOF 32.768ms
011   8    333ns  TOF  21.8ms    1us TOF 65.536ms
100  16    667ns  TOF  43.7ms    2us TOF 131.072ms
101  32   1.33us  TOF  87.4ms    4us TOF 262.144ns
110  64   2.67us  TOF 174.8ms    8us TOF 524.288ms
111 128   5.33us  TOF 349.5ms   16us TOF 1.048576s */

  TC0    = TCNT+50; // first interrupt
}
void main(void) {
  OC_Init0();
  DDRP |= 0x80;  // LED
  asm cli
  for(;;) {
    PTP ^= 0x80; // flash LED
  }
}
```

Question: Can we prove it is interrupting at 100 Hz?
Method: Add scope to an input/output port

Results: Interrupts are happening, but timing is too fast
Action: Switch to run mode, hit reset button
Results: It is running!

2.11.3.5. Instrumentation: dump into array without filtering

a debugger instrument

dumps strategic information into an array at run time.
observe the contents of the array at a later time.
use debugger allows you to visualize when running.

```
short DataBuf[100];
int DataCnt=0;
void Dump(short data){
  if(DataCnt==100){
    return;
  }
  DataBuf[DataCnt] = data;   // save
  DataCnt++;
}
```

2.11.3.6. Instrumentation: dump into array with filtering.

A filter is a software/hardware condition that must be true in order to place data into the array.

```
  if(Happiness==VERY){
     Dump(stuff);
  }
```

Instrusiveness

2.11.3.7. Monitor using the LED display

A monitor is an independent output process

executes very fast, so is minimally intrusive
small amounts of strategic information

Examples

LCD display
LED's on individual otherwise unused output bits

2.11.4. Performance Debugging

- verification of timing behavior of our system
- a dynamic process
  - system is run, and
  - dynamic behavior compared to expected results

2.11.4.1. Instrumentation with independent counter

```
unsigned short Tbuf[100];
unsigned short Tcnt=0;
void RecordTime(void){
  if(Tcnt==100){
    return;
  }
  Tbuf[Tcnt] = TCNT;   // current time
  Tcnt++;
}
```

2.11.4.2. Instrumentation Output Port.

add to ritual

```
  DDRT  |= 0x01;   // PT0 connected to scope
```

put at beginning of function

```
  PTT |= 0x01;    // PT0=1
  PTT_PTT0 = 1;   // PT0=1
```

put at end of function

```
  PTT &= ~0x01;   // PT0=0
  PTT_PTT0 = 0;   // PT0=0
```

2.11.4.3. Measurement of Dynamic Efficiency

measure dynamic efficiency of our software.

1) count bus cycles using the assembly listing

too hard

2) uses an internal timer called TCNT.

```
unsigned short before,elasped;
void main(void){
  ss = 100;
  before = TCNT;
  tt = sqrt(ss);
  elapsed = TCNT-before-12;
}
```
Program 2.27: Empirical measurement of dynamic efficiency.

3) use an oscilloscope or a logic analyzer.
```
void main(void){
  DDRT |= 0x80;  // PT7 connected to scope
  ss = 100;
  while(1){
    PTT_PTT7 = 1;  // set PT7 high
    tt = sqrt(ss);
    PTT_PTT7 = 0;  // clear PT7 low
  }
}
```
Program 2.29. Empirical measurement of dynamic efficiency.

2.11.5. Profiling
collects the time history of strategic variables

```
unsigned short time[100];   // when
unsigned short place[100];  // where
unsigned short data[100];   // what
unsigned short n;
void profile(unsigned short thePlace,
             unsigned short theData){
  if(n==100){
    return;
  }
  time[n]  = TCNT; // record current time
  place[n] = thePlace;
  data[n]  = theData;
  n++;
}

unsigned short sqrt(unsigned short s){
  unsigned short t,oldt;
  t = 0;        // secant method
profile(0,t);
  if(s>0){
profile(1,t);
     t = 32;   // initial guess 2.0
     do{
profile(2,t);
       oldt = t;  // from the last
       t = ((t*t+16*s)/t)/2;}
     while(t!=oldt);
  }
profile(3,t);
  return t;
}
```
Program 2.30: A profile dumping into array.

2.11.5.2. Profiling using an Output Port

```
unsigned short sqrt(unsigned short s){
unsigned short t,oldt;
PTT = 0;
  t = 0;        // secant method
  if(s>0){
PTT = 1;
     t = 32;    // initial guess 2.0
     do{
PTT = 2;
       oldt = t;  // from the last
       t = ((t*t+16*s)/t)/2;
     }
     while(t!=oldt);
  }
PTT = 3;
  return t;
}
```

Bottom line

Buy and use debugging tools
Understand overhead of the debugging tool
Dumps, Monitors, Profiling