Debugging Tools

Software verification is a difficult but important phase.

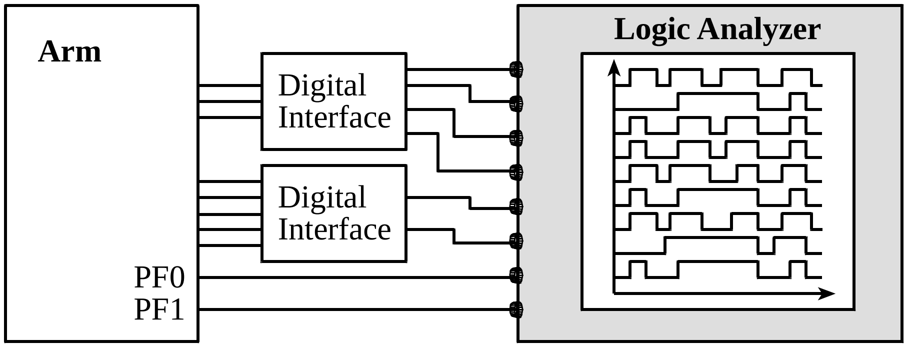
Objectives

- **Debugging hardware**
- **Stabilization**
- **Minimally intrusive debugging instruments**
- **Profiling**
- **White box versus black box testing**

Control and observability

A **logic analyzer** is a multiple channel digital storage scope

- numerous digital signals at various points in time
- attached to strategic I/O signals, real-time measurement
- attached to heart beats, profile execution
- massive amount of information
- triggering to capture data at appropriate times
- must interpret the data
- nonintrusive
- good for real time observation of I/O signals



A logic analyzer and example output.
Show the digilentinc analog discovery
www.**digilentinc**.com/Products/Detail.cfm?Prod=**ANALOG-DISCOVERY**

Events that are observable in real time
- The input and output signals of the system
- Dumps (record in real time, observe later off line)
- Extra output pins

Software-based debuggers
- breakpoint by replacing the instruction with a trap
- can not be performed when the software is in ROM
- Single step with periodic interrupts

Write debugging information into flash so you can analyze systems even if power is lost.

***Hardware based debuggers* (JTag).**
- exists on the microcomputer chip itself
- communicates with the debugging computer
- ability to observe software execution in real time,
- hardware support to set breakpoints,
- the ability to stop the computer and
- supports hardware breakpoints.
- memory and I/O ports are accessible while running

Debugging Theory

“rough and ready” manual methods

desk-checking,
dumps, and
print statements

Debugging instrument
software code that is added for the purpose of debugging.

***stabilize* the system**

creating a test routine that fixes (or stabilizes) all inputs.
can reproduce the exact inputs over and over again.
modify the program,
change in our outputs is a function of modification
and not due to a change in the input parameters.

non-intrusive/intrusive Intrusiveness is used as a measure of the degree of perturbation caused in program performance by an instrument.

Develop your own unique debugging style.

- place all print statements in a unique column
- specific pattern in their names.
- test a run time global flag
  leaves a copy of the code in the final system
  simplifies “on-site” customer support.
- Use conditional compilation
  performance and effectiveness.

Black box
Just inputs/outputs
Know what it does but not how it works
Have pin numbers/signal names on the connector

White box testing
Can probe inside
Know both what it does and how it works
Have internal schematics

Functional debugging

verification of input/output parameters
a static process where
inputs are supplied,
the system is run, and
the outputs are compared against expected results.

Single Stepping or Trace

Breakpoints without filtering

Instrumentation: print statements
difficulty with print statements in embedded systems
a standard "printer" may not be available.
print statement itself may so slow, *intrusive*.
print hardware used for normal operation
If you wish to use printf, you need to create a **fputc** function like this

```
int fputc(int ch, FILE *f){
  Serial_OutChar(ch);
  return (1);
}
```

```
int fgetc (FILE *f){
  return (Serial_InChar());
}
int ferror(FILE *f){
  /* Your implementation of ferror */
  return EOF;
}
```

Appropriate debugging methods

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 to visualize when running.

```
long DebugList[100];
unsigned int DebugCnt=0;
void RecordIt(long data){
  if(DebugCnt==100){
    return;
  }
  DebugList[DebugCnt]=data;
  DebugCnt++;
}
```

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(condition){
    RecordIt(MyData);
  }
```

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, PF0

```
#define PF0                (*((volatile unsigned long *)0x40025004))
#define GPIO_PORTF_DATA_R  (*((volatile unsigned long *)0x400253FC))
  PF0 = 0x01;
  GPIO_PORTF_DATA_R |= 0x01;
  PF0 = 0x00;
  GPIO_PORTF_DATA_R &= ~0x01;
```

Performance Debugging

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

Instrumentation with independent counter

```
unsigned long Tbuf[100];
unsigned int Tcnt=0;
void RecordTime(void){
  if(Tcnt==100)
    return;
  Tbuf[Tcnt] = NVIC_ST_CURRENT_R;
  // 24-bit SysTick counter, 20ns
  Tcnt++;
}
```

Instrumentation Output Port.

`4804 LDR r0,[pc,#16] ;r0= 0x400063FC`
`6800 LDR r0,[r0,#0x00] ;r0=PORTC`
`F0400020 ORR r0,r0,#0x20 ;set bit 5`
`4903 LDR r1,[pc,#12] ;r1= 0x40006000`
`F8C103FC STR r0,[r1,#0x3FC];write PORTC`
	`GPIO_PORTC_DATA_R |= 0x20;`

This subroutine is nonreentrant because of the read-modify-write access to a global.

Show an example of how this assembly listing is found

Run on board or on simulator

Discuss compiler optimization

```
#define GPIO_PORTC_DATA_R       (*((volatile unsigned long *)0x400063FC))
#define GPIO_PORTC0             (*((volatile unsigned long *)0x40006004))
#define GPIO_PORTC1             (*((volatile unsigned long *)0x40006008))
#define GPIO_PORTC2             (*((volatile unsigned long *)0x40006010))
#define GPIO_PORTC3             (*((volatile unsigned long *)0x40006020))
#define GPIO_PORTC4             (*((volatile unsigned long *)0x40006040))
#define GPIO_PORTC5             (*((volatile unsigned long *)0x40006080))
#define GPIO_PORTC6             (*((volatile unsigned long *)0x40006100))
#define GPIO_PORTC7             (*((volatile unsigned long *)0x40006200))
#define GPIO_PORTC12            (*((volatile unsigned long *)0x40006018))
```

Bit specific addressing

`2020 MOVS r0,#0x20`
`4902 LDR r1,[pc,#8] ;r1=0x40006080`
`6008 STR r0,[r1,#0x00];set bit 5`
	`GPIO_PORTC5 = 0x20;`

This subroutine is reentrant because of the read-modify-write access is atomic.

```
#define GPIO_PORTF_DATA_BITS_R  ((volatile unsigned long *)0x40025000)
#define GPIO_PORTF_DATA_R       (*((volatile unsigned long *)0x400253FC))
```

Create a MACRO for the pin PF1, LED

Show data sheet for TM4F123

Measurement of Dynamic Efficiency

measure dynamic efficiency of our software.

1) count bus cycles using the assembly listing

too hard

Show where to find bus cycle times

http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM4_TRM_r0p1.pdf

2) uses an internal timer called SysTick.

```
unsigned long before,elasped;
// ranges from 0 to NVIC_ST_RELOAD_R
unsigned long OS_Time(void){
  return NVIC_ST_CURRENT_R; // 20ns
}
void main(void){
```

initialize stuff

```
before = OS_Time();
```

software to test, assuming no interrupts

```
 elapsed = OS_TimeDiff(OS_Time(),before);
}
```

Empirical measurement of dynamic efficiency.

3) use an oscilloscope or a logic analyzer.

```
#define GPIO_PORTD_DIR_R   (*((volatile unsigned long *)0x40007400))
#define GPIO_PORTD_DEN_R   (*((volatile unsigned long *)0x4000751C))
#define SYSCTL_RCGC2_R     (*((volatile unsigned long *)0x400FE108))

void main(void){
volatile unsigned long delay;
SYSCTL_RCGC2_R |= 0x08;    // activate D
delay = SYSCTL_RCGC2_R;    // allow time to finish
GPIO_PORTD_DIR_R |= 0x20;  // PD5 debugging output
GPIO_PORTD_DEN_R |= 0x20;  // digital enable
   ss = 100;
   while(1){
     GPIO_PORTD_DATA_R |= 0x20;
     tt = sqrt(ss);
     GPIO_PORTD_DATA_R &= ~0x20;
   }
}
```

Empirical measurement of dynamic efficiency.

Can you do the above with less overhead?

Profiling

Collects the time history of strategic variables
Where executing, and when it is executing
What is the data, and when is the data these values
Where executing, when it is executing, and what is the data

```
unsigned long 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] = OS_Time(); // 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;
}
```

A profile dumping into array.

Profiling using an Output Port

One way to profile is to make output 0,1,2,3,…etc

Another way to profile is to make output 1,2,4,8,…etc

```
unsigned int sqrt(unsigned int s){
unsigned int t,oldt;
  GPIO_PORTC4 = 0x10;
  t=0;       // secant method
  if(s>0) {
     GPIO_PORTC5 = 0x20;
     t=32;   // initial guess 2.0
     do{
       GPIO_PORTC6 = 0x40;
       oldt=t;  // from the last
       t=((t*t+16*s)/t)/2;
       GPIO_PORTC6 = 0;
     }
     while(t!=oldt);
     GPIO_PORTC5 = 0;
     }
  GPIO_PORTC4 = 0;
  return t;
}
```

Thread Profile

One way to profile is to set bit on enter, clear bit on exit

```
    GPIO_PORTC4 = 0x10;
    RxFifo_Put(data); // clears RDRF
    GPIO_PORTC4 = 0;

    GPIO_PORTC5 = 0x20;
    TxFifo_Get(&data)){
    GPIO_PORTC5 = 0;
```

I/O bound or CPU bound??

```
unsigned short TxFifo_Size(void){
  if(TxPutPt<TxGetPt){
    return(TxPutPt+TXFIFOSIZE-TxGetPt);
  }
  else{
    return(TxPutPt-TxGetPt);
  }
}
```

A) Measure fifo size versus time

When is it I/O bound?

When is it CPU bound?

B) Measure a histogram of Fifo sizes

```
unsigned long histogram[TXFIFOSIZE];
void Collect(void){
  histogram[TxFifo_Size()]++;
}
```