Introduction to Embedded Systems

CS/ECE 6780/5780

Al Davis

Today's topics:

- intro to interfacing (non-interrupt style)
- also covered in Chap. 3 of the text

Interface Control

- Critical SW role in ES design
  - ES's distinguished by
    - » large variety of I/O devices
    - » each device is controlled by software
      - method is device specific
      - but there are general strategies (covered subsequently)
- I/O interfaces
  - physical connections
  - software routines that affect information exchange

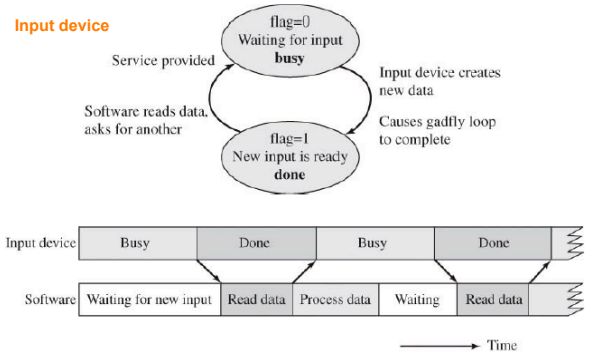
Interface Performance Measures

- Latency
  - delay between service request and service completion
    - » includes both software and hardware delays
  - for real-time systems
    - » guarantee must be made for worst-case latency
- Bandwidth (or throughput)
  - maximum rate at which data can be processed
- Priority
  - determines service order when more than one request is pending

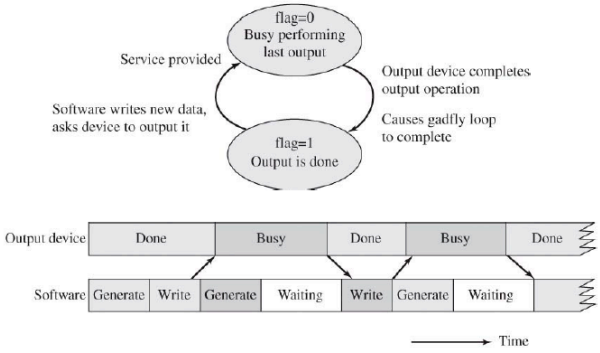
Synchronizing SW w/ I/O Devices

- Problem: I/O devices operate in parallel w/ controller
  - pro: parallelism enhances performance
  - con: it's hard for humans to get it right
- Hardware common case
  - 3 states: idle, busy, or done
  - when not idle
    - » busy and done alternate
- I/O or CPU bound (unbuffered vs. buffered interfaces)
  - I/O bound is typical
    - » I/O devices often much slower than controller SW loop
      - synchronization is required
      - unbuffered interface works but SW has to do significant babysitting
        - we'll start with this more typical case
  - CPU bound
    - » still need synchronization for accurate information transfer
    - » buffering required to store I/O transactions

Polling (a.k.a. Gadfly) Loop Example



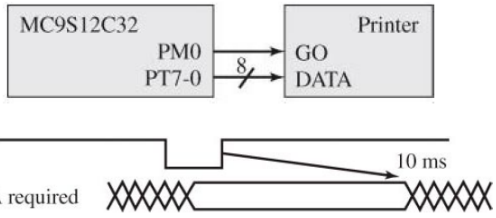
Gadfly: Output Device



Synchronization Mechanisms

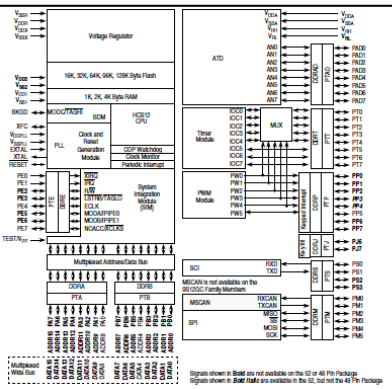
- **Blind cycle**
  - **SW waits a fixed amount of time for the I/O to complete**
    - » **then samples input or produces another output**
- **Gadfly (busy waiting)**
  - **check I/O status flag once per iteration (previous example)**
    - » **waits for flag to indicate I/O done state**
- **Interrupt**
  - **I/O requests SW to become active**
- **Periodic polling**
  - **timer based interrupt requests software activity**
    - » **6812 TCNT timer both more accurate and energy efficient than a cycle counting software timer**
- **Direct Memory Access (DMA)**
  - **I/O device transfers data to/from controller memory**
    - » **memory used as a mailbox to facilitate communication**

Blind Cycle Printer Interface



Note implicit timing assumptions:
printer can **always** take a new byte every 10ms
pulse width is long enough for printer to see it in **all** cases
protocol: data is valid on rising edge of GO signal

Remember the Device



Port data and direction registers

Initialize and Output to a Printer

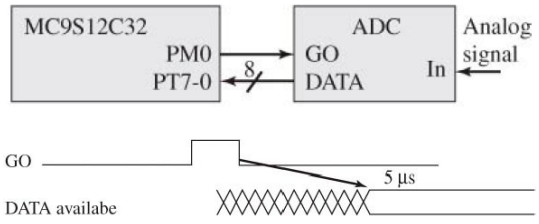
Blind Cycle Method

DDR = data direction register ports T or M both outputs here

```
void Init(void){
  DDRT = 0xFF;   // outputs
  DDRM|= 0x01;
  PTM |= 1;      // GO=1
  Timer_Init();}
void Out(unsigned char value){
  PTT = value;
  PTM&=~0x01;  // GO=0
  PTM|=0x01;   // GO=1
  Timer_MsWait(10);}  // 10ms
```

What should be added to this code?

Blind Cycle ADC Interface



Note different GO sense and implicit timing assumption

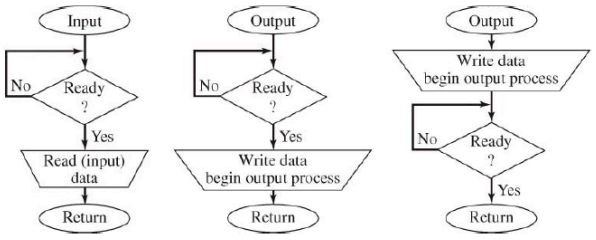
Initialize and Read ADC

```
void Init(void){
  DDRT = 0x00;   // input DATA
  DDRM|= 0x01;
  PTM &=~0x01;   // GO=0
  Timer_Init();}
unsigned char In(void){
  PTM |= 0x01;    // GO=1
  PTM &=~0x01;    // GO=0
  Timer_UsWait(5);
  return(PTT);}
```

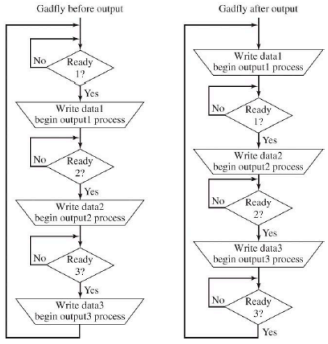
Blind Cycle Evaluation

- **Advantages**
  - simple and predictable
- **Problems**
  - inflexible – periodic timing assumption
  - inefficient if the delay is long
    - » problematic if other parallel threads compete
      - might interfere with periodicity assumption
    - » If done wrong
      - faster CPU upgrade may break code
        - e.g. pulse width will be smaller
- **Works well for simple, high-speed devices**
  - fact: most ES's have a few blind cycle interfaces

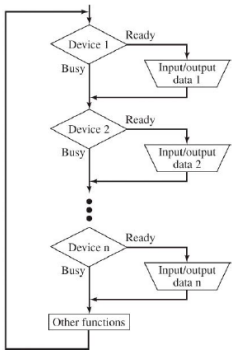
Gadfly Synchronization



Multiple Gadfly Outputs: Single Loop



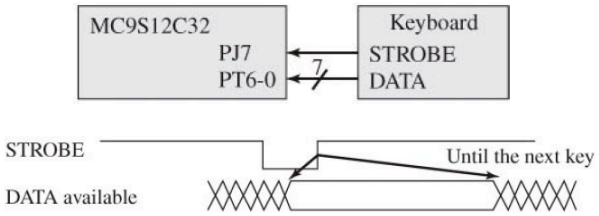
Multiple Gadfly Inputs & Outputs



Gadfly Evaluation

- **Advantages**
  - simple
  - more flexible than blind cycle
- **Potential "gotcha's"**
  - inefficient if devices are not fast
    - » due to long wait times
  - potentially unpredictable
    - » wait until the I/O device is ready
      - → if wait is not known then loop time can't be known
      - if wait is known then use blind cycle model
- **Use with caution**
  - however a common tactic in many ES's
    - » bizarre but it happens even in real time systems
      - in this case code is even more implicit
      - updates become truly scary

Gadfly Keyboard Interface Using Latched Input

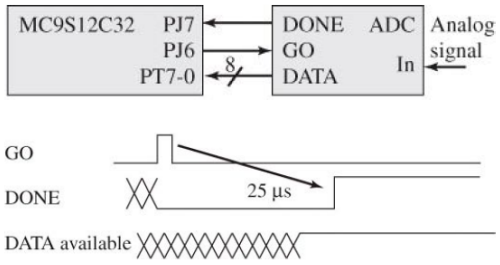


Implicit Timing Assumptions?

Initialize and Read from a Keyboard

```
void Init(void){ // PJ7=STROBE
  DDRJ = 0x00;   // PT6-0 DATA
  DDRT = 0x80;   // PT7 unused output
  PPSJ = 0x80;   // rise on PJ7
  PIFJ= 0x80;}   // clear flag7
unsigned char In(void){
  while((PIFJ&0x80)==0); // wait
  PIFJ = 0x80;  // clear flag7
  return(PTT);
}
```


Gadfly ADC Interface for a Simple Input

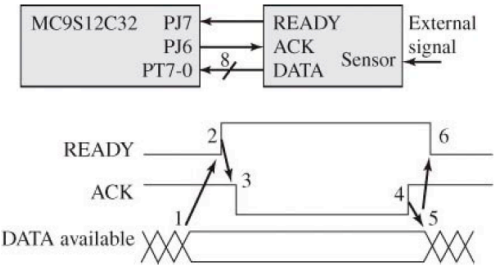


Initialize and Read from an ADC

```
void Init(void){ // PJ7=DONE in
  DDRJ = 0x40;   // PJ6=GO out
  PPSJ = 0x80;   // rise on PJ7
  DDRT = 0x00;   // PT7-0 DATA in
  PTJ &=~0x40;}  // GO=0
unsigned char In(void){
  PIFJ = 0x80;   // clear flag7
  PTJ |= 0x40;   // GO=1
  PTJ &=~0x40;   // GO=0
  while((PIFJ&0x80)==0);
  return(PTT);}
```


Gadfly External Sensor Interface

Using an *overlapping* variant of a 4 cycle input handshake



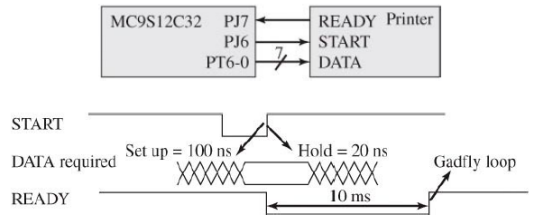
implicit timing assumptions?

Initialize and Read from a Sensor

```
void Init(void){// PJ7=READY in
  DDRJ = 0x40;  // PJ6=ACK out
  PPSJ = 0x80;  // rise on PJ7
  DDRT = 0x00;  // PT7-0 DATA in
  PIFJ = 0x80;  // clear flag7
  PTJ |= 0x40;} // ACK=1
unsigned char In(void){
  unsigned char data;
  while((PIFJ&0x80)==0);
  PTJ &=~0x40; // ACK=0
  data = PTT;  // read data
  PIFJ = 0x80; // clear flag7
  PTJ |=0x40;  // ACK=1
  return(data);}
```


Gadfly Printer Interface Using Output Handshake

Non-overlapping 4-cycle variant



Notice anything weird here?

Initialize and Write to a Printer

```
void Init(void){// PJ7=READY in
  DDRJ = 0x40;  // PJ6=START out
  PPSJ = 0x80;  // rise on PJ7
  DDRT = 0xFF;  // PT7-0 DATA out
  PTJ |= 0x40;} // START=1
void Out(unsigned char data){
  PIFJ = 0x80;  // clear flag
  PTJ &=~0x40;  // START=0
  PTT  = data;  // write data
  PTJ |= 0x40;  // START=1
  while((PIFJ&0x80)==0);}
```

Is this code robust – if not why not?

Gadfly Synch to Digital Thermometer

- Dallas Semiconductor DS1620
  - range -55 to 125 C with .5 C resolution
  - data encoded using 9 bit 2's complement values
    - » → basis weights
      - -128, 64, 32, 16, 8, 4, 2, 1, 0.5

Temperature	Binary Value	Hex Value
+125.0°	011111010	$0FA
+64.0°	010000000	$080
0.5°	000000001	$001
0°	000000000	$000
-0.5°	111111111	$1FF
-16.0°	111100000	$1E0
-55.0°	110010010	$192

More DS1650

- Can also be used as a thermostat
  - 2 registers
    - » TL – threshold low
    - » TH – threshold high
  - temp >> TH then TH_{output} goes to +5 v
  - temp << TL then TL_{output} goes to +5 v
- Interface is serial (big banging) – using 3 I/O's
  - RST is reset
  - rising CLK samples DQ
    - » DQ is bidirectional
      - output to DS1650 is a command
        - sent as 8 bit burst
      - Input from DS1650 is also an 8 bit serial burst
  - status flags
    - » Done: 1→valid conversion, 0→ in progress
    - » THF: 1 → temp above TH
    - » TLF: 1 → temp below TL

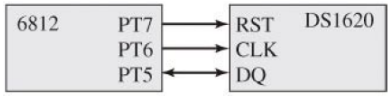
Interface Code for DS1650

- Mind numbingly boring to cover in class
  - so we won't
    - » advice: go through text section 3.5.7 to get the general idea
    - » details will become apparent in the labs
- Serial interfaces are common
  - DS1650 is just an example
  - pins are expensive
    - » In both $ and Joules
      - pad drivers consume area ($) and energy (p or nJ)
  - hence many cheap sensors rely on serial interfaces
    - » often with industry standard protocols and interfaces
    - » e.g. SPI and SCI
      - serial peripheral/communications interface
    - » 6812 has direct support for both
      - we'll use them in the labs

Concluding Remarks

- Interfaces are the center of mass for ES control
  - diversity of peripheral circuitry → diverse control styles
    - » this was just an introduction
- Beware implicit timing assumptions
  - make them as explicit as possible in your code
  - even if this only makes sense via a comment
    - » defining constants is even better
      - blind cycle → timer delay
      - gadfly loop → iteration count & instruction timing
        - beware lack of portability w/ faster or slower clock speeds
- Lab 2 is happening this week
  - it's not as assembler intensive as initially planned
    - » due to student and TA feedback
  - clearly I'm a rookie at teaching this class
    - » let's all hope this condition gets better

Initialization




```
void DS_Init(void){ // PT7=RST=0
  DDRT = 0xE0;    // PT6=CLK=1
  PTT = 0x60;}    // PT5=DQ=1
```