Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

Unit 6: Interfacing to Sensors and Actuators (Ch. 9)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

Connecting the Analog and Digital Worlds

Cyber:

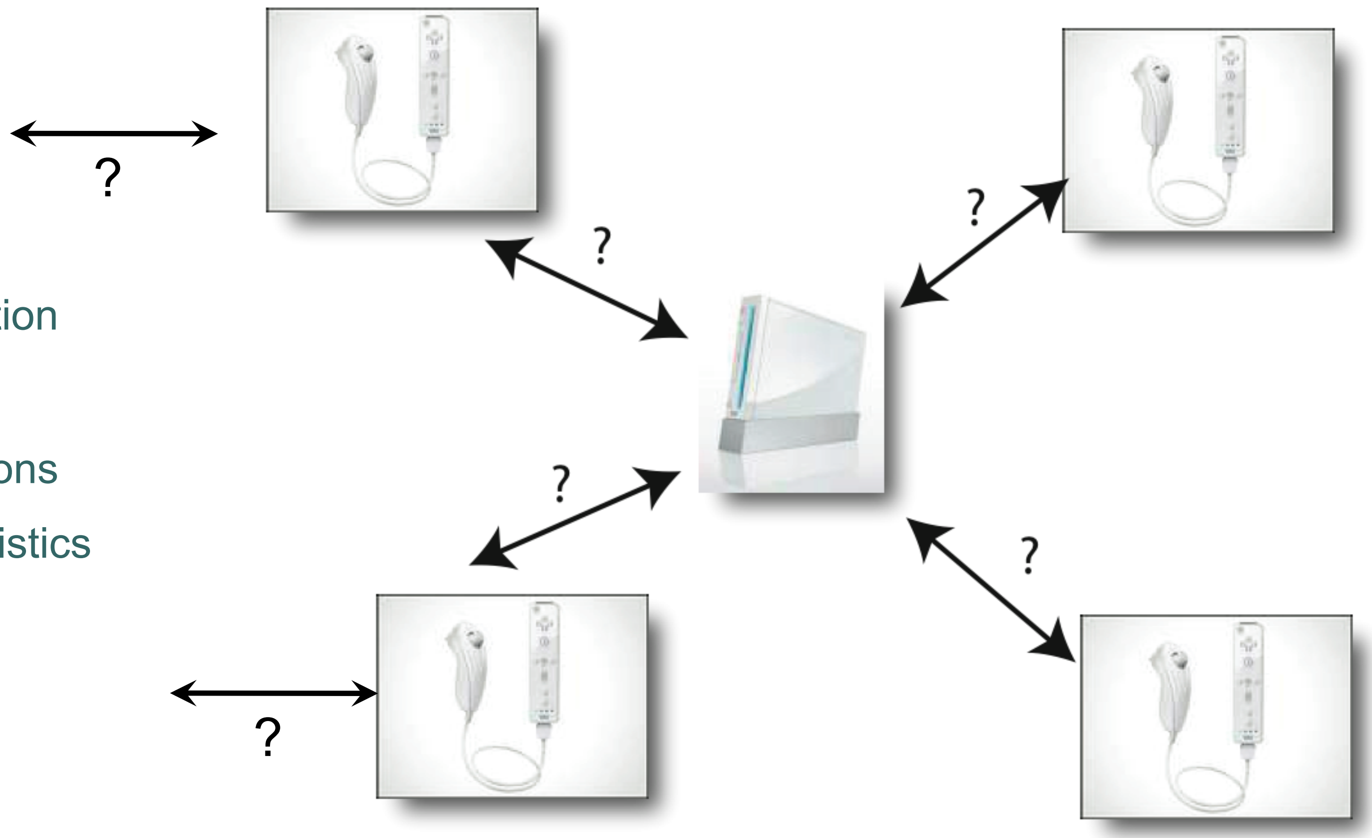
- Digital
- Discrete in time
- Sequential

Physical:

- Continuum
- Continuous in time
- Concurrent

What is going on here?

Interfaces



- Physical Connection
- Handshake
- Multiple connections
- Timing Characteristics

A Typical Microcomputer Board

This board has analog and digital inputs and outputs. What are they? How do they work?

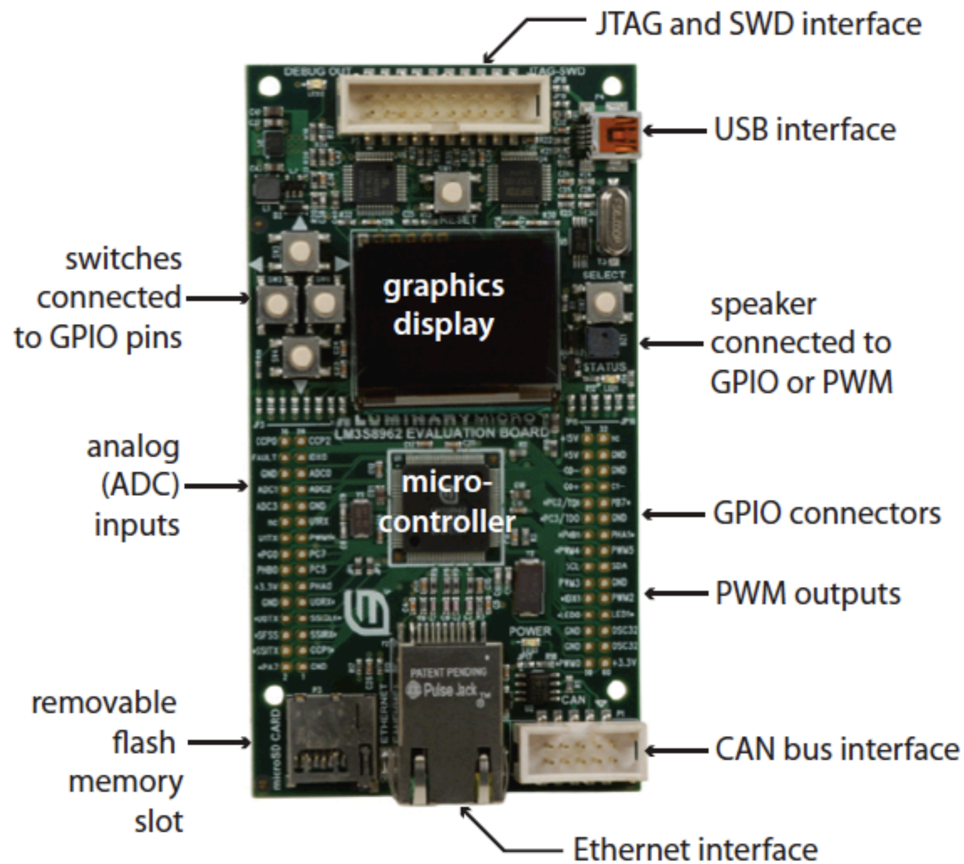
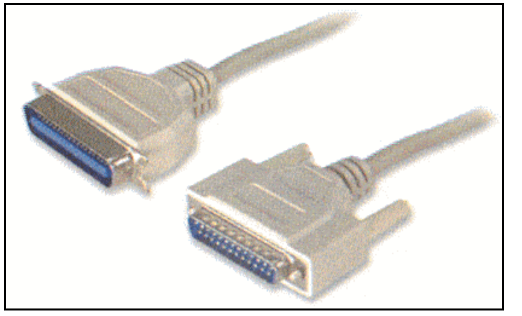


Figure 8.1: Stellaris®LM3S8962 evaluation board (Luminary Micro®, 2008a).

Parallel vs. Serial Digital Interfaces

- Parallel
  - Multiple data lines transmitting data
  - Speed
  - Ex: PCI, ATA, CF cards, Bus



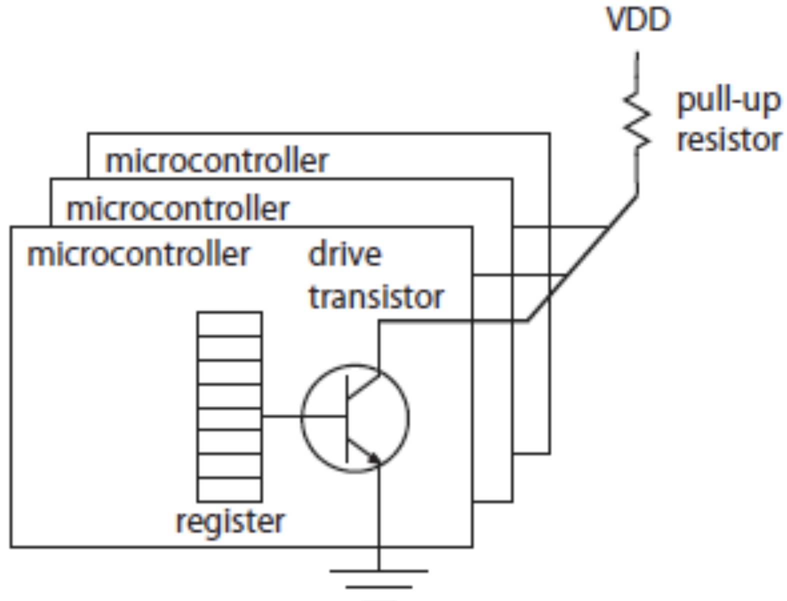
- Serial
  - Single data line transmitting data
  - Low Power, length
  - Ex: USB, SATA, SD cards, PCI-Express

Simple Digital Output: GPIO

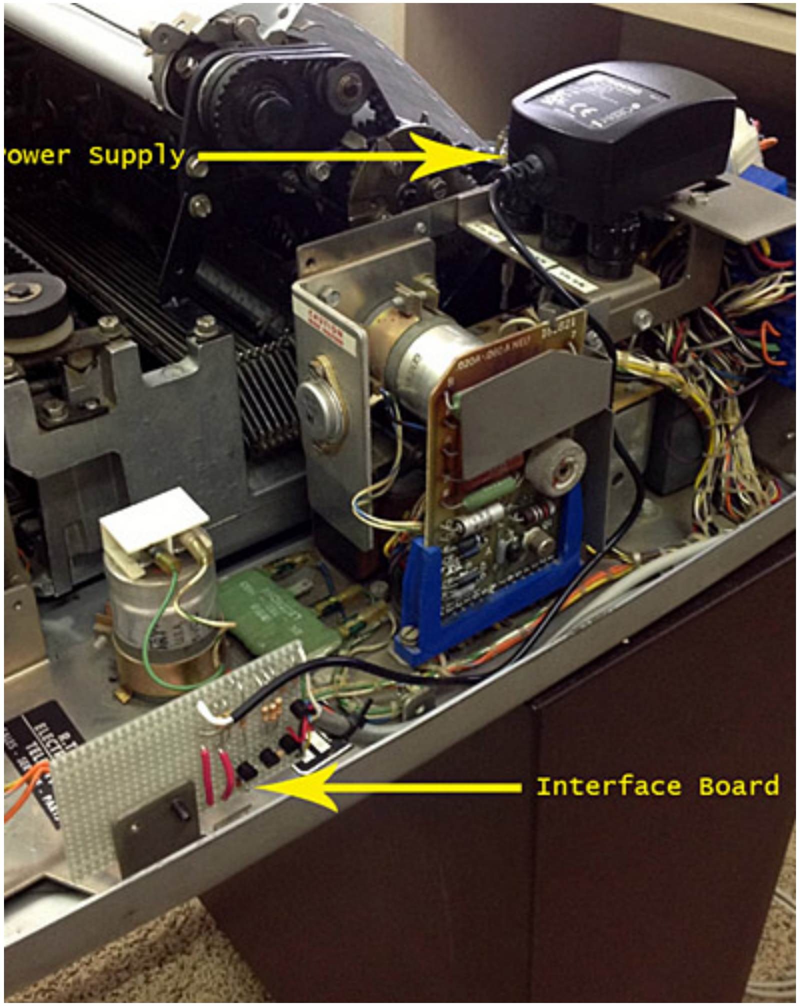
Open collector circuits are often used on GPIO (general-purpose I/O) pins of a microcontroller.

The same pin can be used for input and output. And multiple users can connect to the same bus.

Why is the current limited?



Serial Interfaces – RS-232

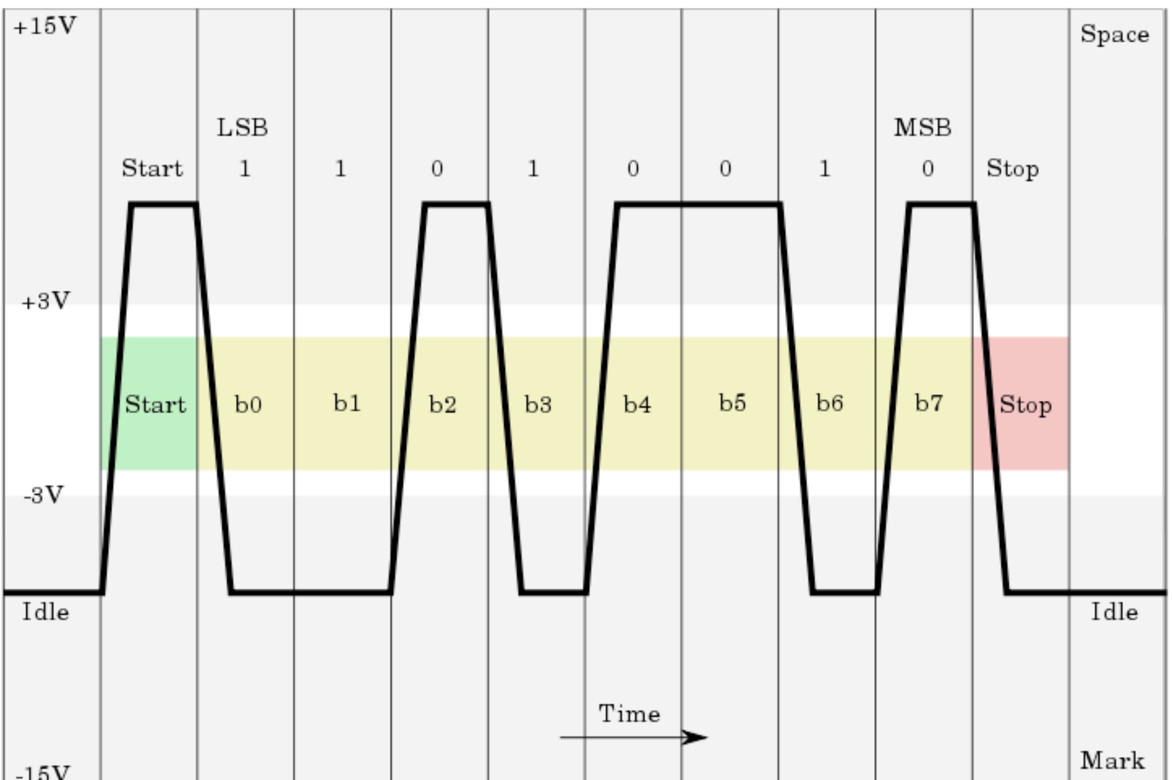


Serial Interfaces

The old but persistent RS-232 standard supports asynchronous serial connections (no common clock).

How does it work?

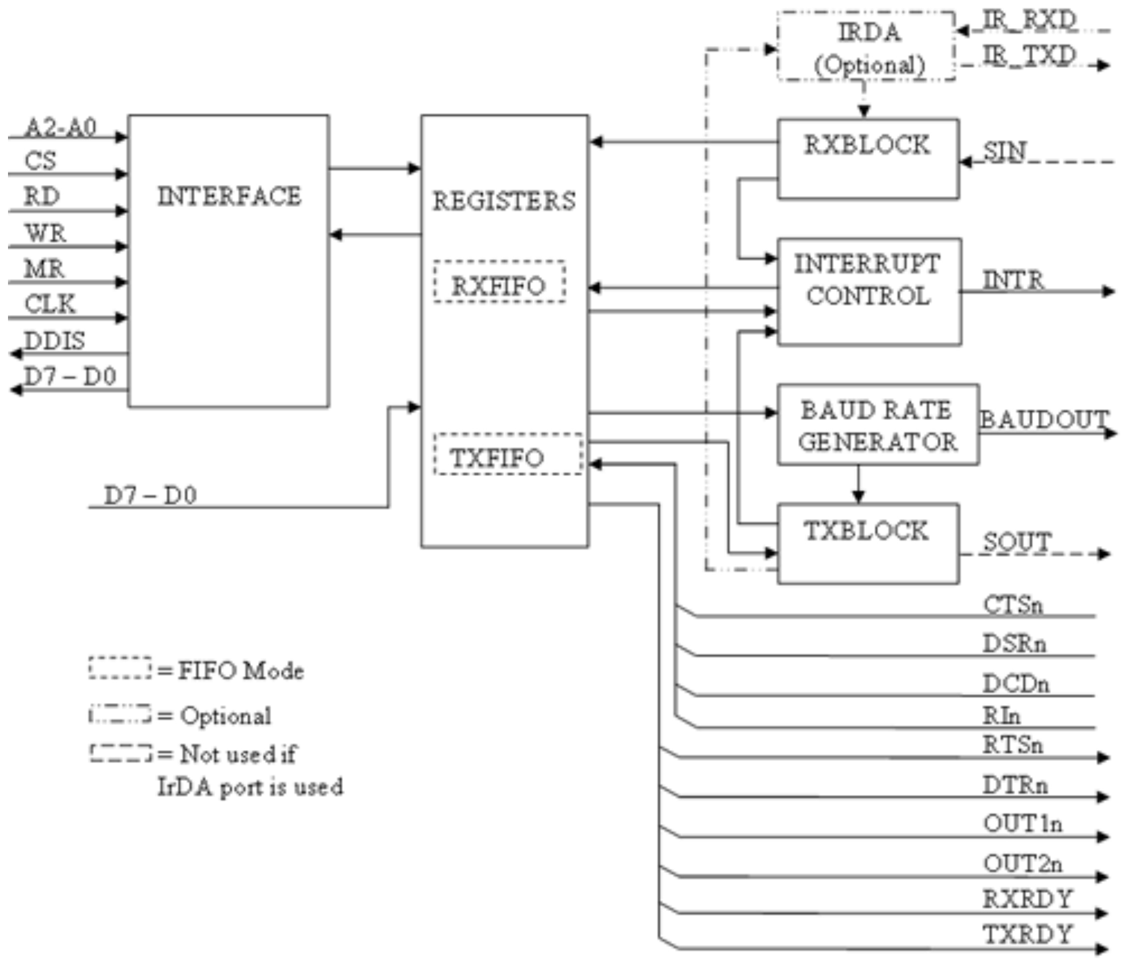
Many but not all uses of RS-232 are being replaced by USB, which is electrical simpler but with a more complex protocol.



Uppercase ASCII "K" character (0x4b) with 1 start bit, 8 data bits, 1 stop bit.

UART: Universal Asynchronous Receiver-Transmitter

- Convert serial data to parallel data, and vise versa.
- Uses shift registers to load store data
- Can raise interrupt when data is ready
- Commonly used with RS-232 interface



Example Using a Serial Interface

In an Atmel AVR 8-bit microcontroller, to send a byte over a serial port, the following C code will do:

```
while(!(UCSR0A & 0x20));
UDR0 = x;
```

- x is a variable of type uint8.
- UCSR0A and UDR0 are variables defined in header.
- They refer to memory-mapped registers in the UART.

Send a Sequence of Bytes

```
for(i = 0; i < 8; i++) {
    while(!(UCSR0A & 0x20));
    UDR0 = x[i];
}
```

How long will this take to execute? Assume:

- 57600 baud serial speed.
- 8/57600 =139 microseconds.
- Processor operates at 18 MHz.

Each for loop will consume 2500 cycles.

Receiving via UART

Again, on an Atmel AVR:

```
while(!(UCSR0A & 0x80));
return UDR0;
```

- Wait until the UART has received an incoming byte.
- *The programmer must ensure there will be one!*
- If reading a sequence of bytes, how long will this take?

Under the same assumptions as before, it will take 2500 cycles to receive each byte.

Input Mechanisms in Software

- Polling
  - Main loop checks each I/O device periodically.
  - If input is ready, processor initiates communication.

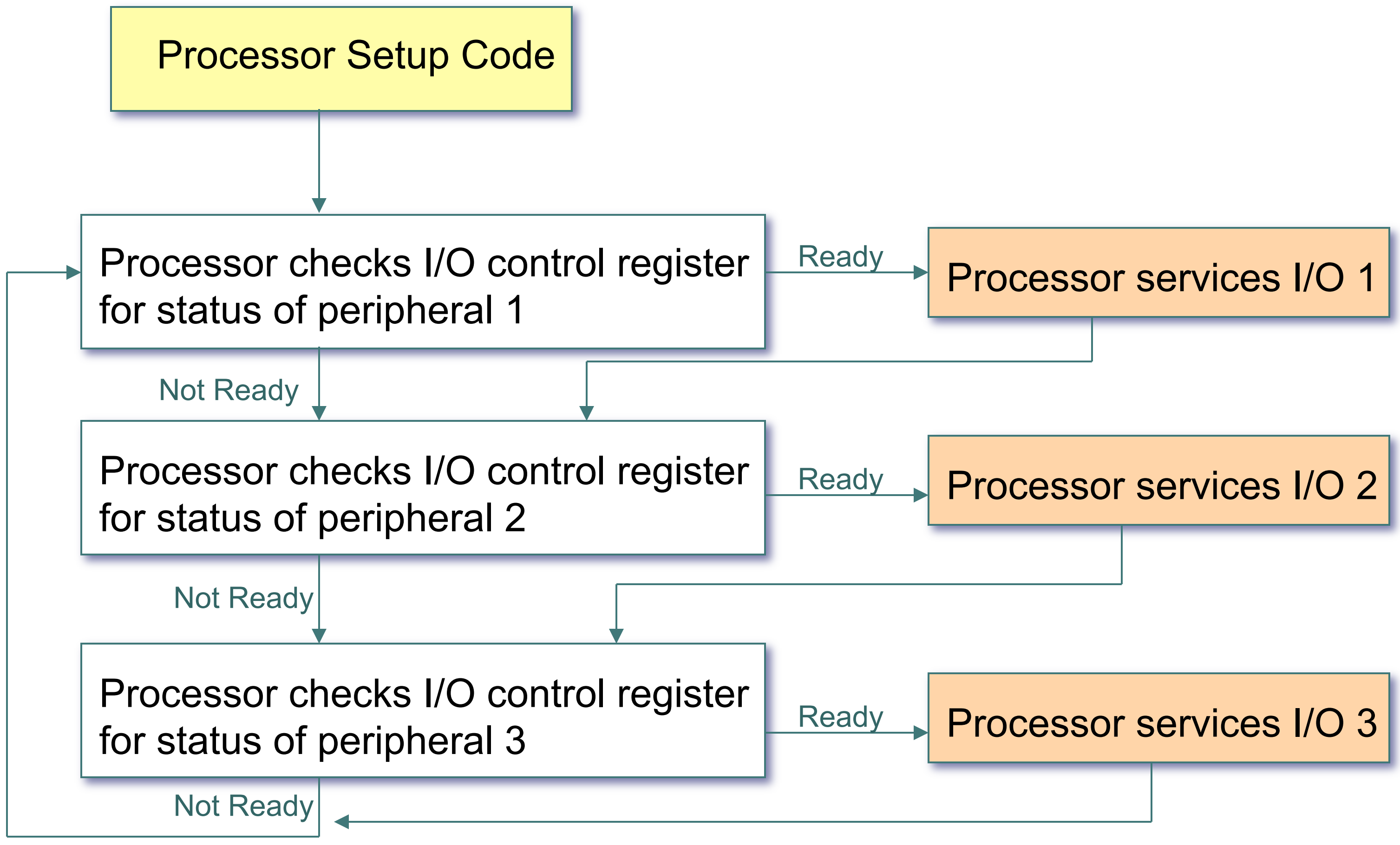
- Interrupts
  - External hardware alerts the processor that input is ready.
  - Processor suspends what it is doing.
  - Processor invokes an interrupt service routine (ISR).
  - ISR interacts with the application concurrently.

ISR Memory Locations

Table 3-4: **Interrupt and Exception Handling**

On	Hardware jumps to	Software Labels
Start / Reset	0x0	_start
User exception	0x8	_exception_handler
Interrupt	0x10	_interrupt_handler
Break (HW/SW)	0x18	-
Hardware exception	0x20	_hw_exception_handler
Reserved by Xilinx for future use	0x28 - 0x4F	-

Polling

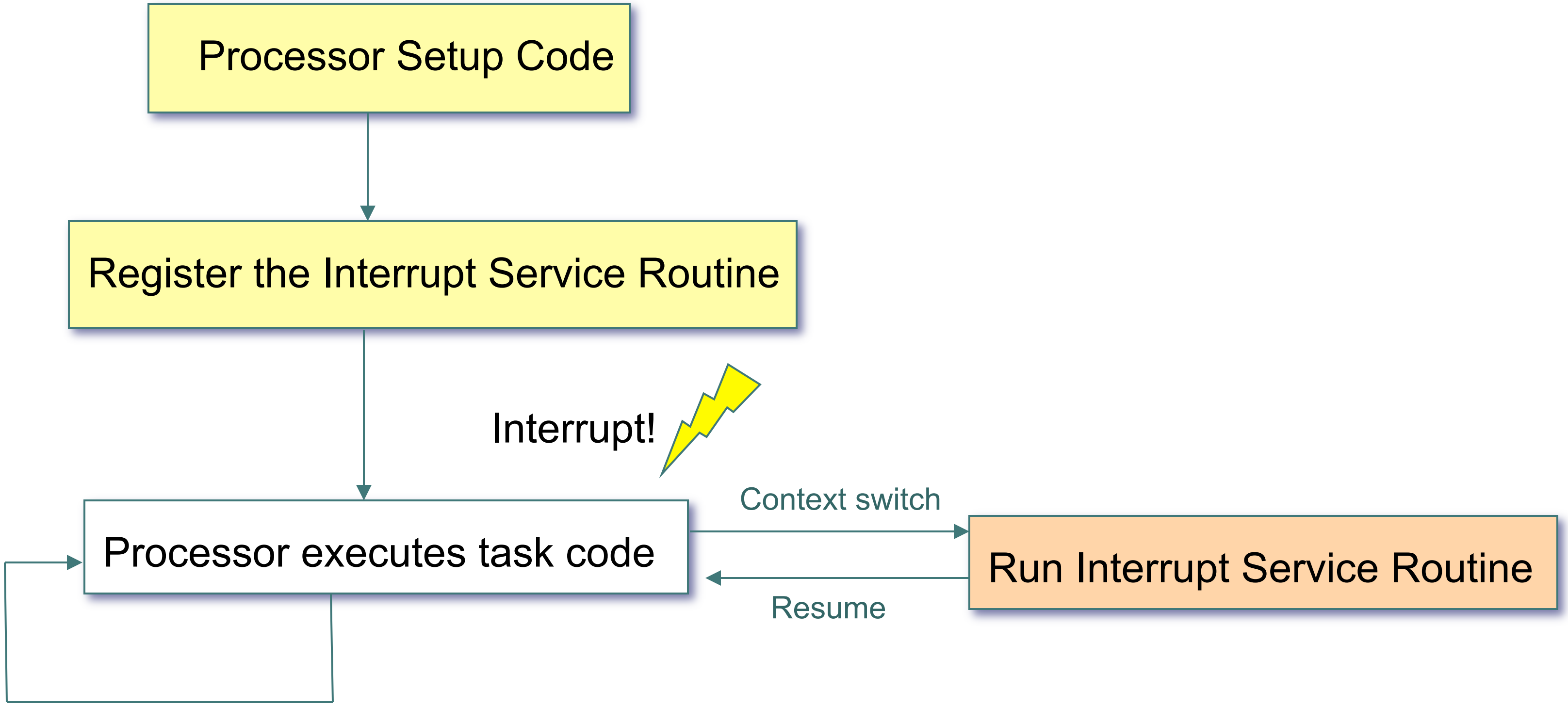


Processor Setup Code
Processor checks I/O control register for status of peripheral 1
Ready
Processor services I/O 1
Not Ready
Processor checks I/O control register for status of peripheral 2
Ready
Processor services I/O 2
Not Ready
Processor checks I/O control register for status of peripheral 3
Ready
Processor services I/O 3
Not Ready

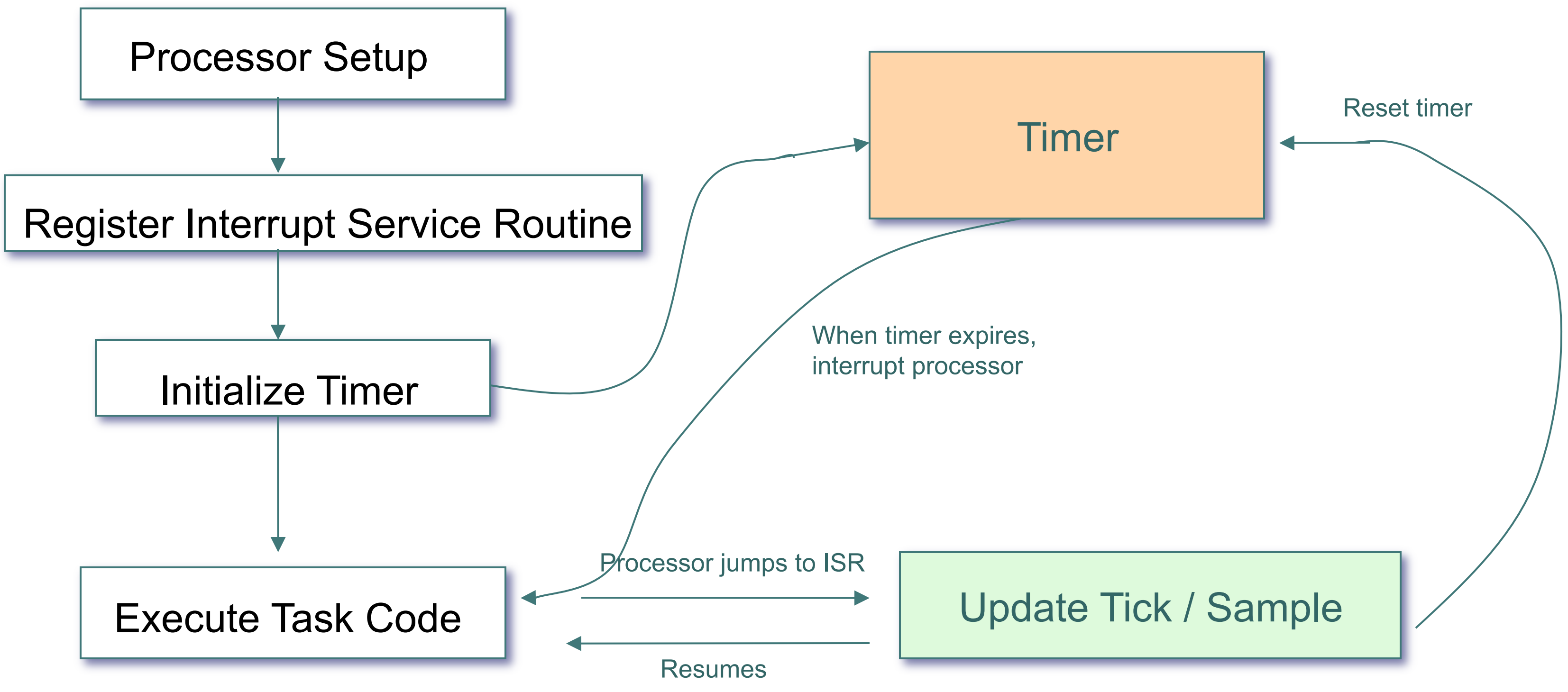
Interrupts

- Interrupt Service Routine

  Short subroutine that handles the interrupt



Timed Interrupt



Issues to Watch For

- Interrupt service routine execution time
- Context switch time
- Nesting of higher priority interrupts
- Interactions between ISR and the application
- Interactions between ISRs
- …

Interrupt example

```
static int iTemperatures[2];
void interrupt vReadTemperatures (void)
{
        iTemperatures[0] = //Read value from hardware 1
        iTemperatures[1] = //Read value from hardware 2
}
void main(void)
{
        int iTemp0, iTemp1;
        //Setup code

        while(TRUE)
        {
                iTemp0 = iTemperatures[0];
                iTemp1 = iTemperatures[1];
                if ( iTemp0 != iTemp1 )
                        // Set off alarm!
        }
}
```

Compiler can optimize this!!

What if interrupt updated both values here?

Shared Data

- Data consistency
  - Critical Section
    - Need to protect portion of the code from other access
    - Disable interrupts
  - Compiler Optimizations
    - Various optimization techniques
    - Volatile keyword, or turn off optimizations

Interrupt example revisited

```
static volatile int iTemperatures[2];
void interrupt vReadTemperatures (void)
{
        iTemperatures[0] = //Read value from hardware 1
        iTemperatures[1] = //Read value from hardware 2
}
void main(void)
{
        int iTemp0, iTemp1;
        //Setup code

        while(TRUE)
        {
                disableInterrupts();
                iTemp0 = iTemperatures[0];
                iTemp1 = iTemperatures[1];
                enableInterrupts();
                if ( iTemp0 != iTemp1 )
                        // Set off alarm!
        }
}
```

```c
#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
#define _SFR_IO8(io_addr) _MMIO_BYTE((io_addr) + 0x20)
#define _SFR_MEM8(mem_addr) _MMIO_BYTE(mem_addr)
#define _BV(bit) (1 << (bit))
```

```c
//Enable interrupts (interrupt.h)
# define sei()  __asm__ __volatile__ ("sei" ::)
//Disable interrupts (interrupt.h)
# define cli()  __asm__ __volatile__ ("cli" ::)
#define SIGNAL(signame)                    \
void signame (void) __attribute__ ((signal)); \
void signame (void)
```

```c
//Timer defines (iomx8.h)
#define TCCR1A  _SFR_MEM8 (0x80)
#define TCCR1B  _SFR_MEM8 (0x81)
/* TCCR1B */
#define WGM12   3
#define CS12    2
```

SEI		Global Interrupt Enable
CLI		Global Interrupt Disable

```c
void initialize(void)
{
  cli();

  // Set I/O pins
  DDRB = 0x10;
  PORTB = 0xCF;
  …….

  // Set up timer 1 to generate an interrupt every 1 ms
  TCCR1A = 0x00;
  TCCR1B = (_BV(WGM12) | _BV(CS12));
  OCR1A = 71;
  TIMSK1 = _BV(OCIE1A);

  // Set up the serial port with rx interrupt
  …….

  // Turn on interrupts
  sei();
}
```

```c
// Global variables
volatile uint16_t timer_cnt = 0;
volatile uint8_t timer_on = 0;

// Timer 1 interrupt to time delays in ms
SIGNAL(SIG_OUTPUT_COMPARE1A)
{
  if(timer_cnt)
    timer_cnt--;
  else
    timer_on = 0;
}
```

```c
void delayMs(uint16_t time_ms)
{
  timer_on = 1;
  timer_cnt = time_ms;
  while(timer_on) ;
}
```

iRobot Drive example

```
SIGNAL(SIG_USART_RECV) {
  uint8_t temp;
  temp = UDR0;

  if(sensors_flag) {
    sensors_in[sensors_index++] = temp;
    if(sensors_index >= Sen6Size)
      sensors_flag = 0;
  }
}
```

```
void delayAndUpdateSensors(uint16_t time_ms){
  uint8_t temp;

  timer_on = 1;
  timer_cnt = time_ms;
  while(timer_on) {
    if(!sensors_flag){
      for(temp = 0; temp < Sen6Size; temp++)
        sensors[temp] = sensors_in[temp];
      // Update running totals of distance and angle

      byteTx(CmdSensors);
      byteTx(6);
      sensors_index = 0;
      sensors_flag = 1;
    }
  }
}
```

```
for(;;) {
  delayAndUpdateSensors(10);
  if(UserButtonPressed)
  {
   // Drive around until a button or unsafe condition is detected
   while(!(UserButtonPressed) && (!sensors[SenCliffL])
&& (!sensors[SenCliffFL]) && (!sensors[SenCliffFR])
&& (!sensors[SenCliffR])&& (!sensors[SenChAvailable])){
    // Keep turning until the specified angle is reached
    if(turning)
    {  // Code to continue turning }

     // Check for a bump
    else if(sensors[SenBumpDrop] & BumpEither) {
     // Set the turn parameters and reset the angle
     if(sensors[SenBumpDrop] & BumpLeft)
       turn_dir = 0;
     else
       turn_dir = 1;
      //Command to turn iRobot }
    else {
     // Otherwise, drive straight
     drive(300, RadStraight); }

    // Flash the leds in sequence
    // Update LED State
    // wait a little more than one robot tick for sensors to update
    delayAndUpdateSensors(20);
   } //End while loop
   // Stop driving
   drive(0, RadStraight);
  }
}
```

Standards

- Serial:
  - Synchronous:
    - SPI, I2C, JTAG, USB
  - Asynchronous:
    - RS232

- Parallel:
  - Bus protocols
    - Advanced Technology Attachment (ATA)
    - Peripheral Component Interface (PCI)
    - …

An I/O View of a Microcontroller

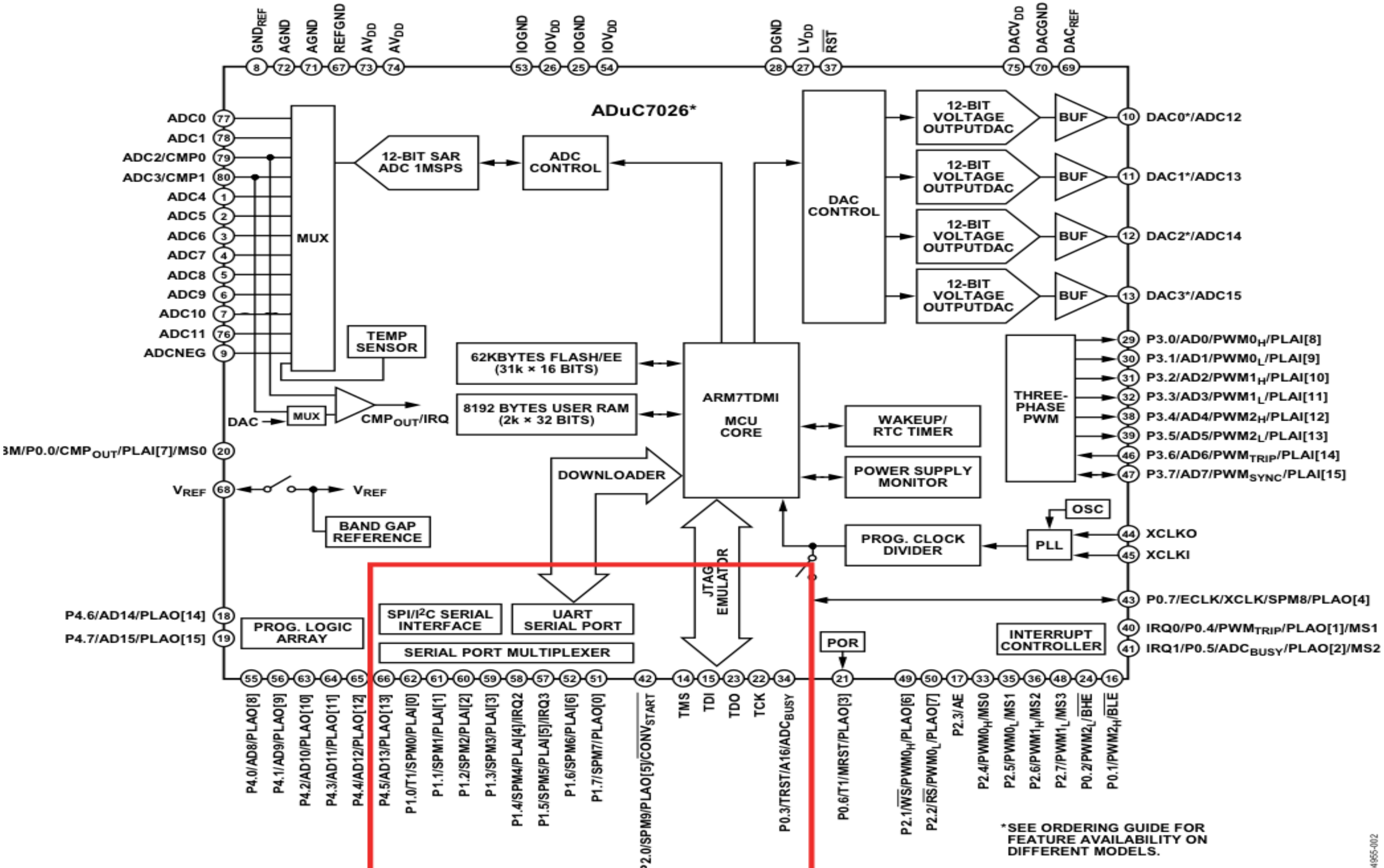
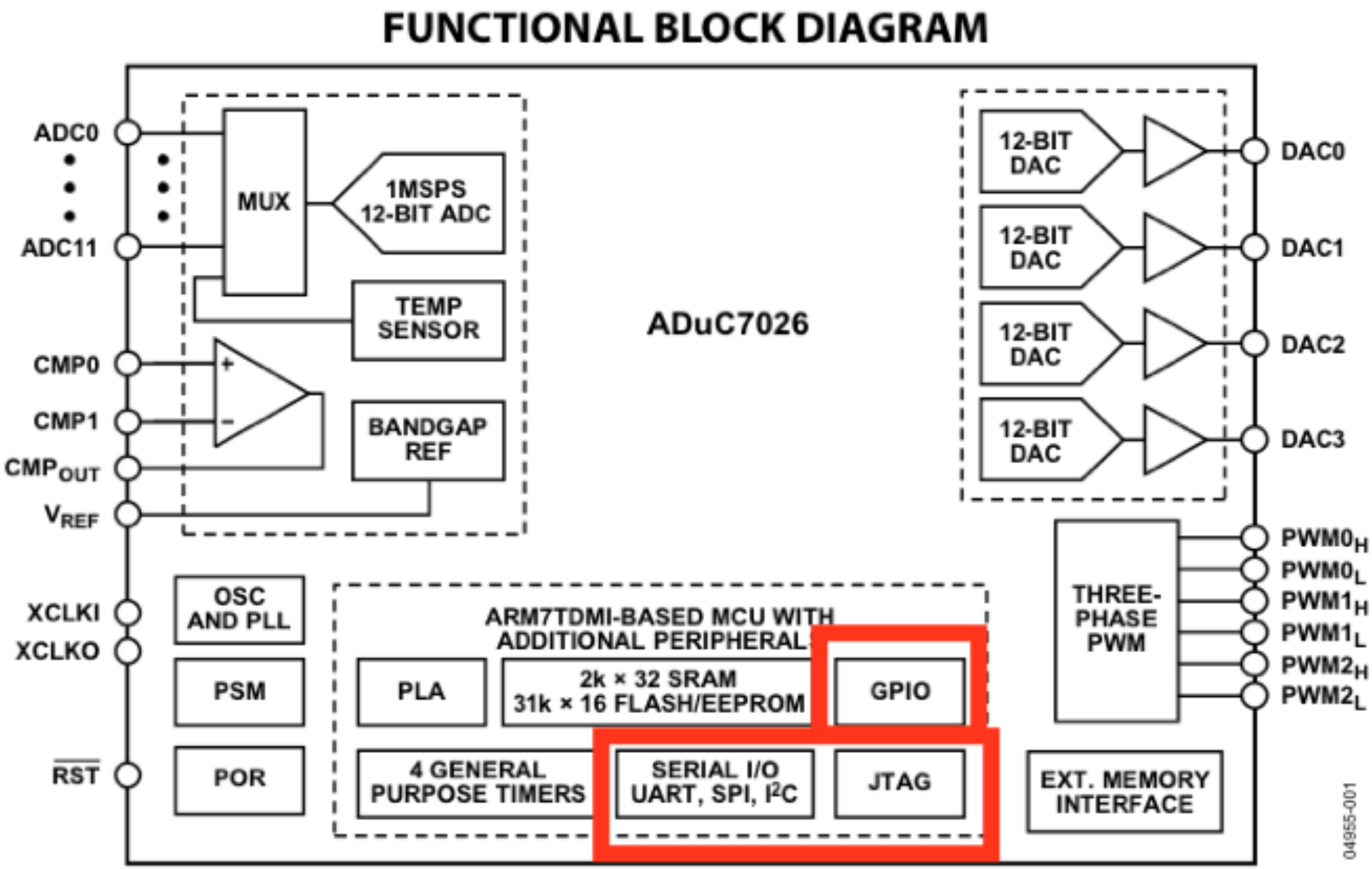


Figure 2.

Another I/O View of a Microcontroller



Application of a Microcontroller

Simple MP3 Player

- Uses serial for control
- Parallel for data transfer
- Direct memory access