Objectives

- **Parallel processing, distributed processing, multithreading**
- **Modular programming**
- **Call graphs,**
- **Flow charts,**
- **Data flow graphs,**
- **Device drivers: serial port,**
- **uVision4 compiler,**
- **Quality software**

Open uart_echo

Draw a call graph

Draw a data flow graph

Open UART2_4F120

Draw a call graph

Draw a data flow graph

Highlight the serial port input and output.

A) How to do decimal input/output?

1) Write your own, like UART2

2) Use **`sprintf`** to create strings then output string

3) Link to Standard library function **`printf()`**,

`Your_putchar` is your implementation that outputs one byte

`fputc _ttywrch` are mapped to **`Your_putchar`**

Standard library function **`getchar()`**

`Your_getchar` is your implementation that input

`fgetc` is mapped to **`Your_getchar`**

Look at the style of `ST7735_4F120`

B) How much driverlib code do you use?

`driverlib` code will have fewer bugs than any you or I write

You will have to certify all code having no critical sections

Most students will want to fit code into 32k

All students must understand everything

Why private versus public?

Information hiding

Reduce coupling

Separate mechanisms from policy

Essence of modular design

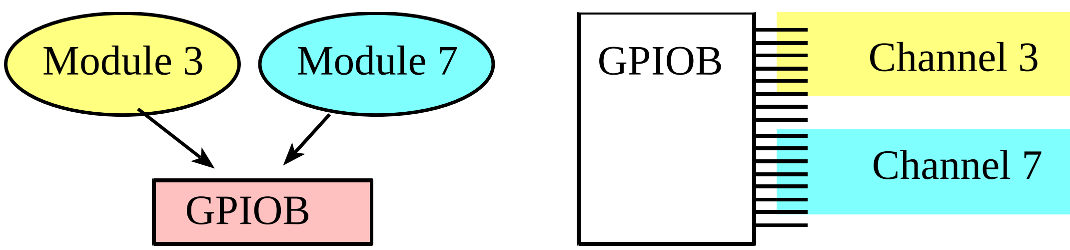
How in C

Public name has *Module Name* and underline

Public object has Prototype in header file

Private globals have `static` modifier

Use call graphs to identify potential conflicts



Flowcharts

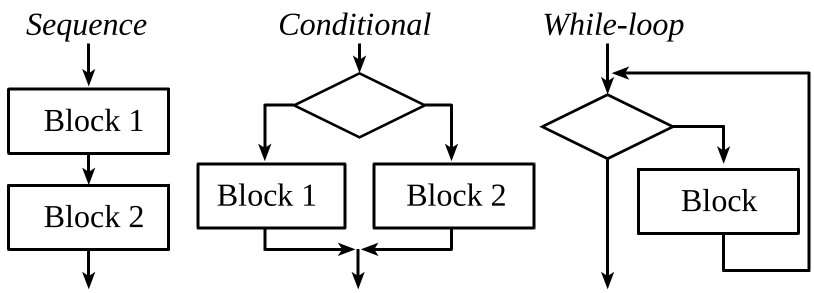


Figure 2.1. Flowchart showing the basic building blocks of structured programming.

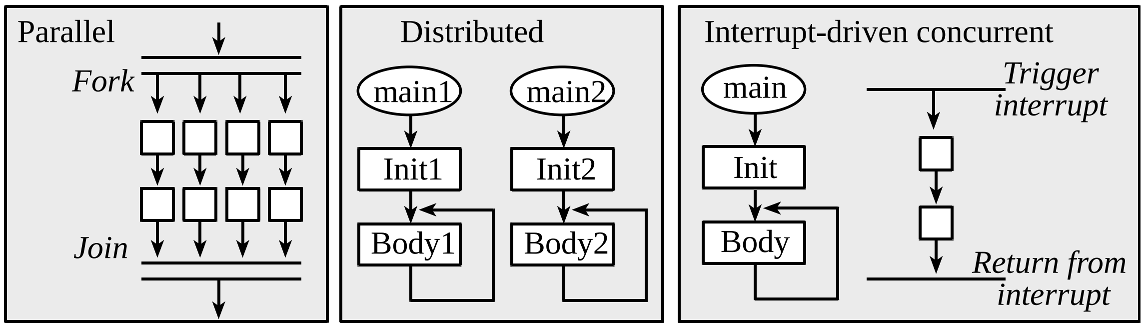


Figure 2.2. Flowchart symbols to describe parallel, distributed, and concurrent programming.

See `FIFO_xxx.zip`

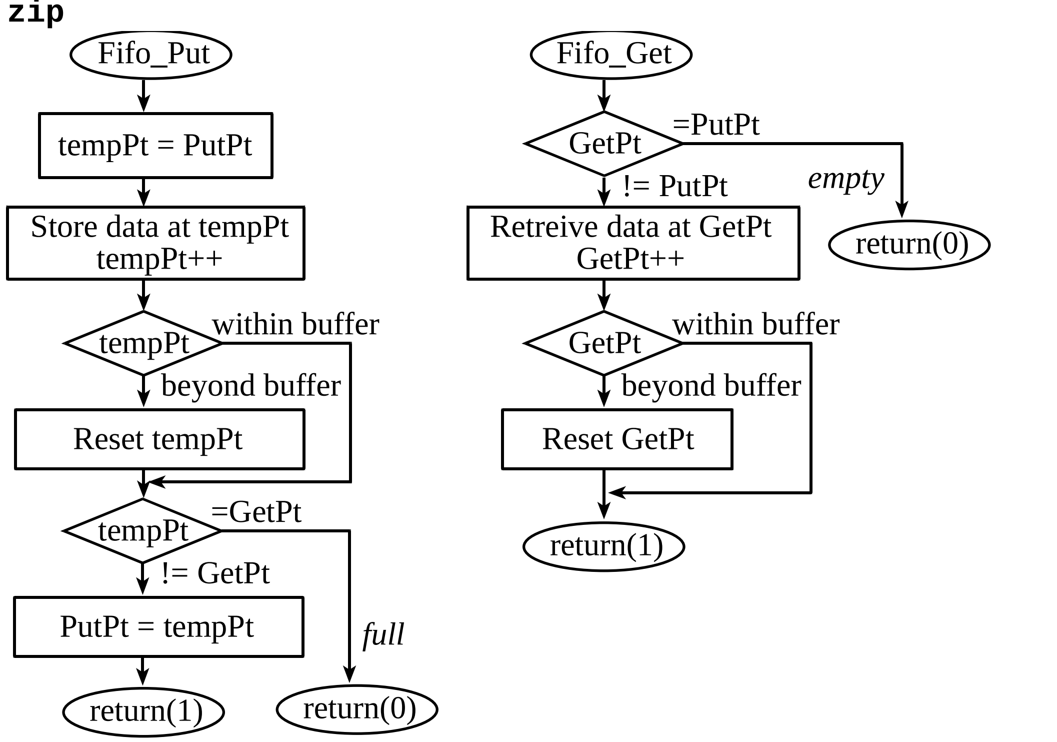


Figure 3.19. Flowcharts of the pointer implementation of the FIFO queue.

```c
// Two-index implementation of the FIFO
// can hold 0 to FIFOSIZE elements
#define FIFOSIZE 16 // must be a power of 2
#define FIFOSUCCESS 1
#define FIFOFAIL    0

typedef char dataType;
unsigned long volatile PutI;  // put next
unsigned long volatile GetI;  // get next
dataType static Fifo[FIFOSIZE];

void Fifo_Init(void){ // this is critical
  // should make atomic
  PutI = GetI = 0;      // Empty
  // end of critical section
}
```

```c
// Two-pointer implementation of the FIFO
// can hold 0 to FIFOSIZE-1 elements
#define FIFOSIZE 16     // can be any size
#define FIFOSUCCESS 1
#define FIFOFAIL    0

typedef char dataType;
dataType volatile *PutPt;  // put next
dataType volatile *GetPt;  // get next
dataType static Fifo[FIFOSIZE];

void Fifo_Init(void){ // this is critical
  // should make atomic
  PutPt = GetPt = &Fifo[0]; // Empty
  // end of critical section
}
```

<pre>// return FIFOSUCCESS if successful
int Fifo_Put(dataType data){
  if((PutI-GetI) & ~(FIFOSIZE-1)){
    return(FIFOFAIL); // Failed, fifo full
  }
  Fifo[PutI&(FIFOSIZE-1)] = data; // put
  PutI++; // Success, update
  return(FIFOSUCCESS);
}
 	
// return FIFOSUCCESS if successful
int Fifo_Get(dataType *datapt){
  if(PutI == GetI){
    return(FIFOFAIL); // Empty if PutI=GetI
  }
  *datapt = Fifo[GetI&(FIFOSIZE-1)];
  GetI++; // Success, update
  return(FIFOSUCCESS);
}
 	
// number of elements currently stored
// 0 to FIFOSIZE-1
unsigned short Fifo_Size(void){
 return ((unsigned short)(PutI-GetI));
}</pre>
	<pre>int Fifo_Put(dataType data){
	dataType volatile *nextPutPt;
	  nextPutPt = PutPt+1;
	  if(nextPutPt ==&Fifo[FIFOSIZE]){
	    nextPutPt = &Fifo[0]; // wrap
	  }
	  if(nextPutPt == GetPt){
	    return(FIFOFAIL); // Failed, fifo full
	  }
	  else{
	    *(PutPt) = data; // Put
	    PutPt = nextPutPt; // Success, update
	    return(FIFOSUCCESS);
	  }
	}
	int Fifo_Get(dataType *datapt){
	  if(PutPt == GetPt){
	    return(FIFOFAIL);// Empty if PutPt=GetPt
	  }
	  else{
	    *datapt = *(GetPt++);
	    if(GetPt==&Fifo[FIFOSIZE]){
	      GetPt = &Fifo[0]; // wrap
	    }
	    return(FIFOSUCCESS);
	  }
	}</pre>

Program 3.3. Two-pointer implementation of a FIFO.

How do you make an object in C?

Polymorphic

Inheritance

Encapsulation

```
#define AddFifo(NAME,SIZE,TYPE, SUCCESS,FAIL) \
unsigned long volatile PutI ## NAME;  \
unsigned long volatile GetI ## NAME;  \
TYPE static Fifo ## NAME [SIZE];      \
void NAME ## Fifo_Init(void){         \
  PutI ## NAME= GetI ## NAME = 0;     \
}                                     \
int NAME ## Fifo_Put (TYPE data){     \
  if(( PutI ## NAME - GetI ## NAME ) & ~(SIZE-1)){  \
    return(FAIL);      \
  }                    \
  Fifo ## NAME[ PutI ## NAME &(SIZE-1)] = data; \
  PutI ## NAME ## ++;  \
  return(SUCCESS);     \
}                      \
int NAME ## Fifo_Get (TYPE *datapt){  \
  if( PutI ## NAME == GetI ## NAME ){ \
    return(FAIL);      \
  }                    \
```

```
  *datapt = Fifo ## NAME[ GetI ## NAME &(SIZE-1)];  \
  GetI ## NAME ## ++;  \
  return(SUCCESS);     \
}
AddFifo(Tx,32,unsigned char, 1,0)
```

Data Flow graphs

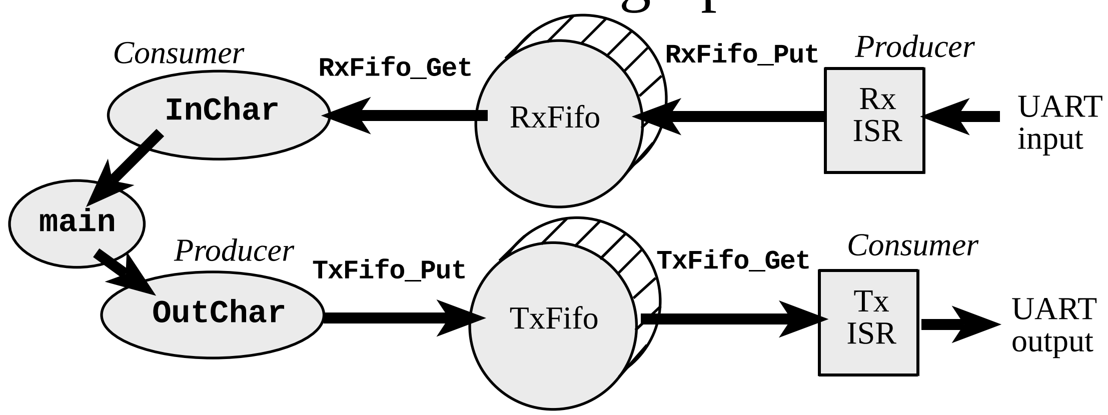
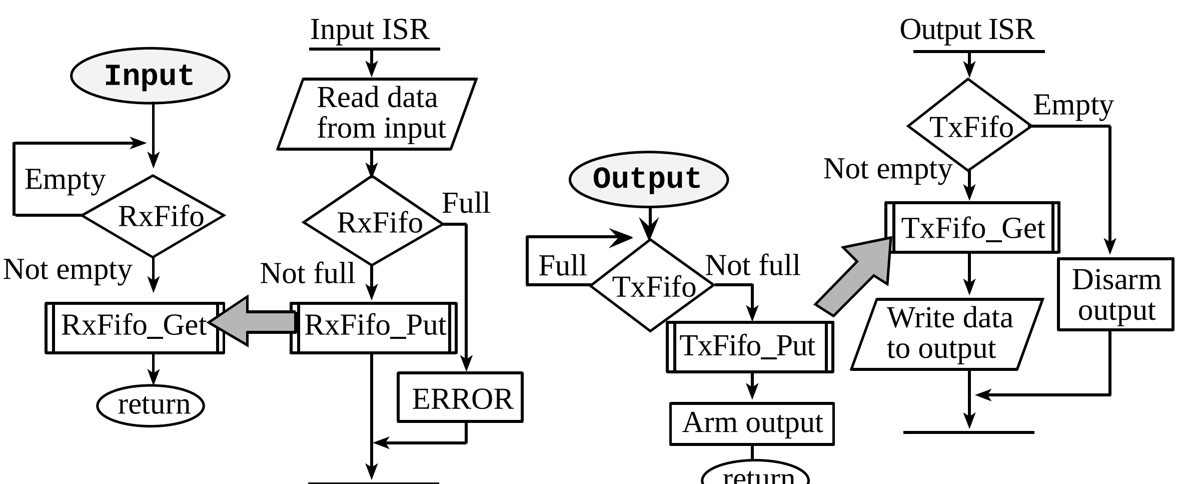


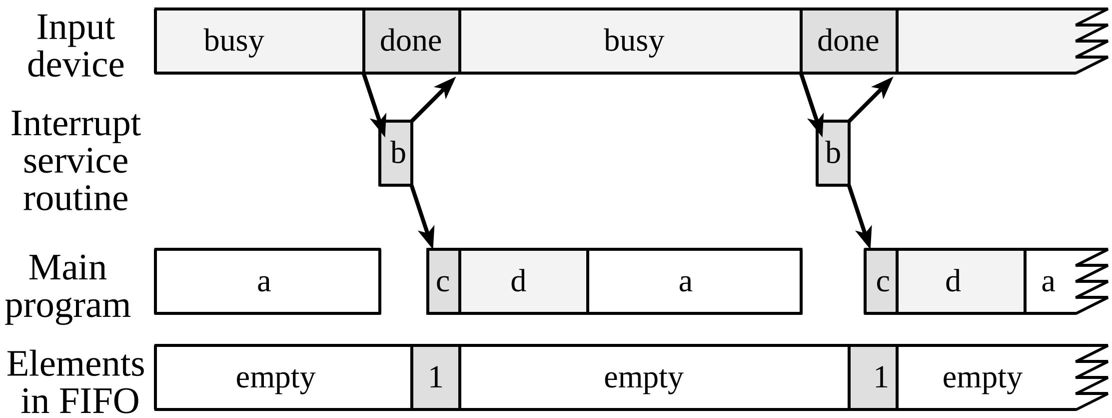
Figure 3.3. A data flow graph showing two FIFOs that buffer data between producers and consumers.

FIFO queues can be used to pass data between threads.



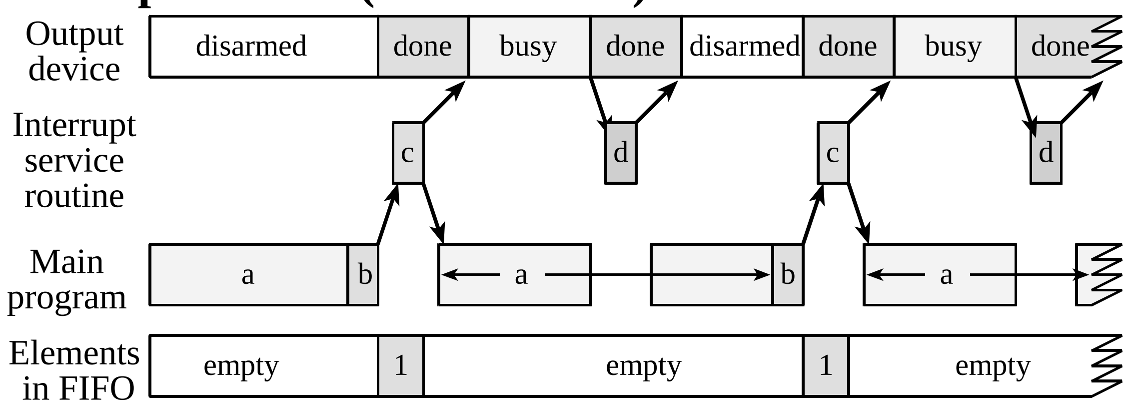
Volume 2 Figure 5.4. In a producer/consumer system, FIFO queues can be used to pass data between threads.

I/O bound input device



Volume 2 Figure 5.6. Hardware/software timing of an I/O bound input interface.

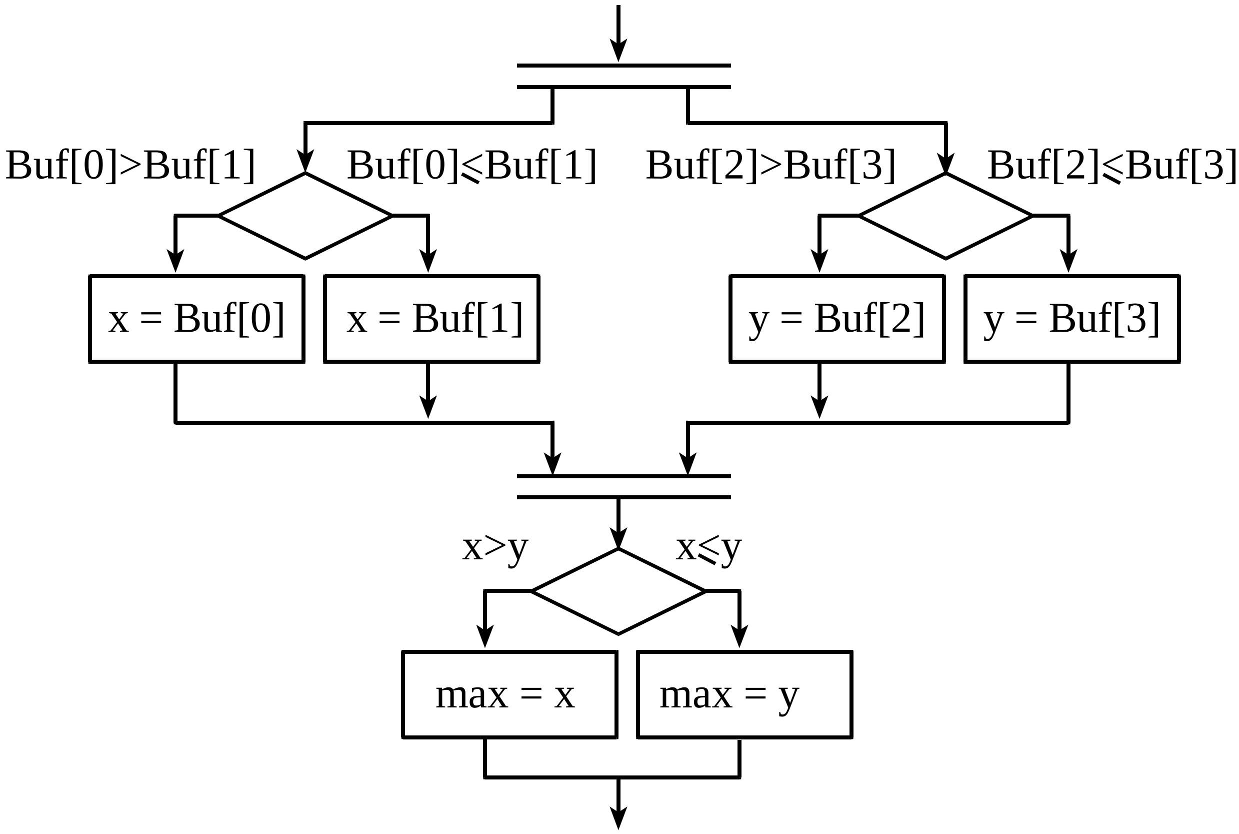
I/O bound output device (buffered I/O)



Volume 2 Figure 5.8. Hardware/software timing of a CPU bound output interface.

Parallel processing:

multiple processors, shared memory
simultaneous execution of two or more software tasks
e.g., multicore Pentium



Distributed processing:

multiple computers, separate memory, I/O network link
simultaneous execution of two or more software tasks
e.g., Lab 6

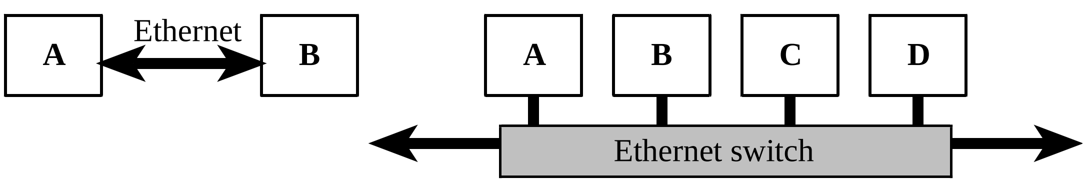


Figure 9.14. Ethernet has a bus-based topology.

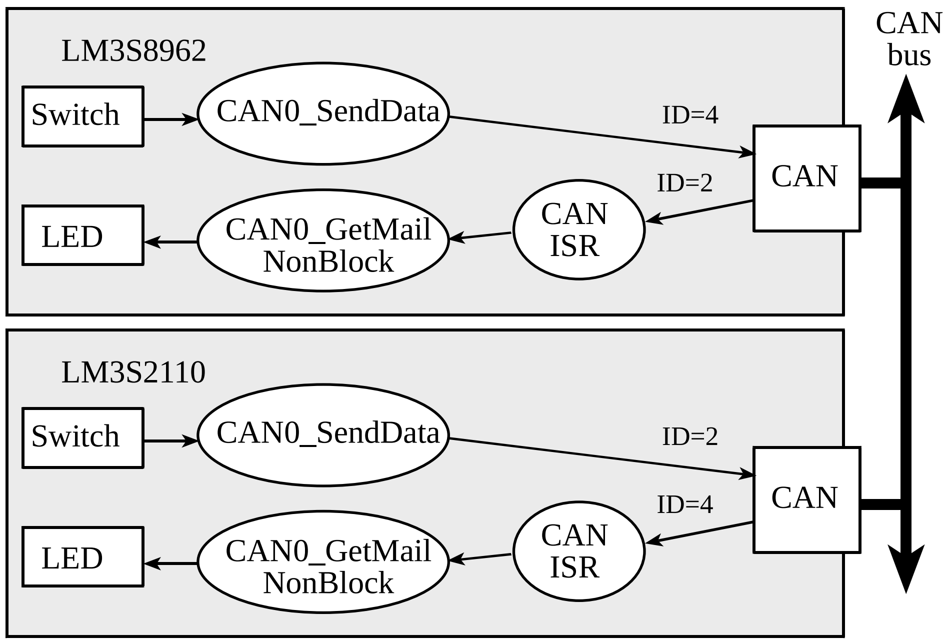
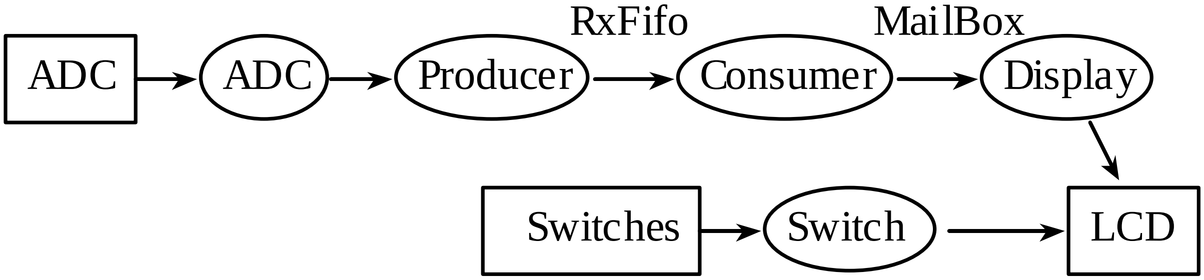


Figure 9.6. Simple CAN network.

Multithreading

One foreground and multiple background threads

Multiple foreground threads using a thread scheduler



If using a LM4F120/TM4C123

0) Use solid wires 22 or 24 gauge wire, attach to bottom

1) female-male connectors (my favorite)

https://www.adafruit.com/products/826

Digi-Key H1505-ND, Hirose DF11-2428SCA

If using a LM3S8962, there are some options

0) Two/four right angle connectors like the LM3S1968

TSW-115-08-L-S-RA

TSW-115-09-L-S-RE

1) Solder solid wire to pins as you need them (repair when needed)

2) One or two female headers

SD-115-G-2 (could use two for LM3S8962 Board)

SD-109-G-2 (could use one for LM3S2110 Board)

SD-107-G-2 (could use two for LM3S2110 Board)

SD-110-G-2 (could use one for LM3S2110 Board)

3) Male headers and female-male connectors (my favorite)

https://www.adafruit.com/products/826

Digi-Key H1505-ND, Hirose DF11-2428SCA

See Course Description page for latest information

SamTec http://www.samtec.com/

Analog Devices http://www.analog.com/en/index.html

Maxim http://www.maxim-ic.com/

Texas Instruments http://www.ti.com

Recap

Call graph

Data flow graph

Flow chart

Fifo queue, buffered I/O

Public versus private