

Real-Time Embedded Systems

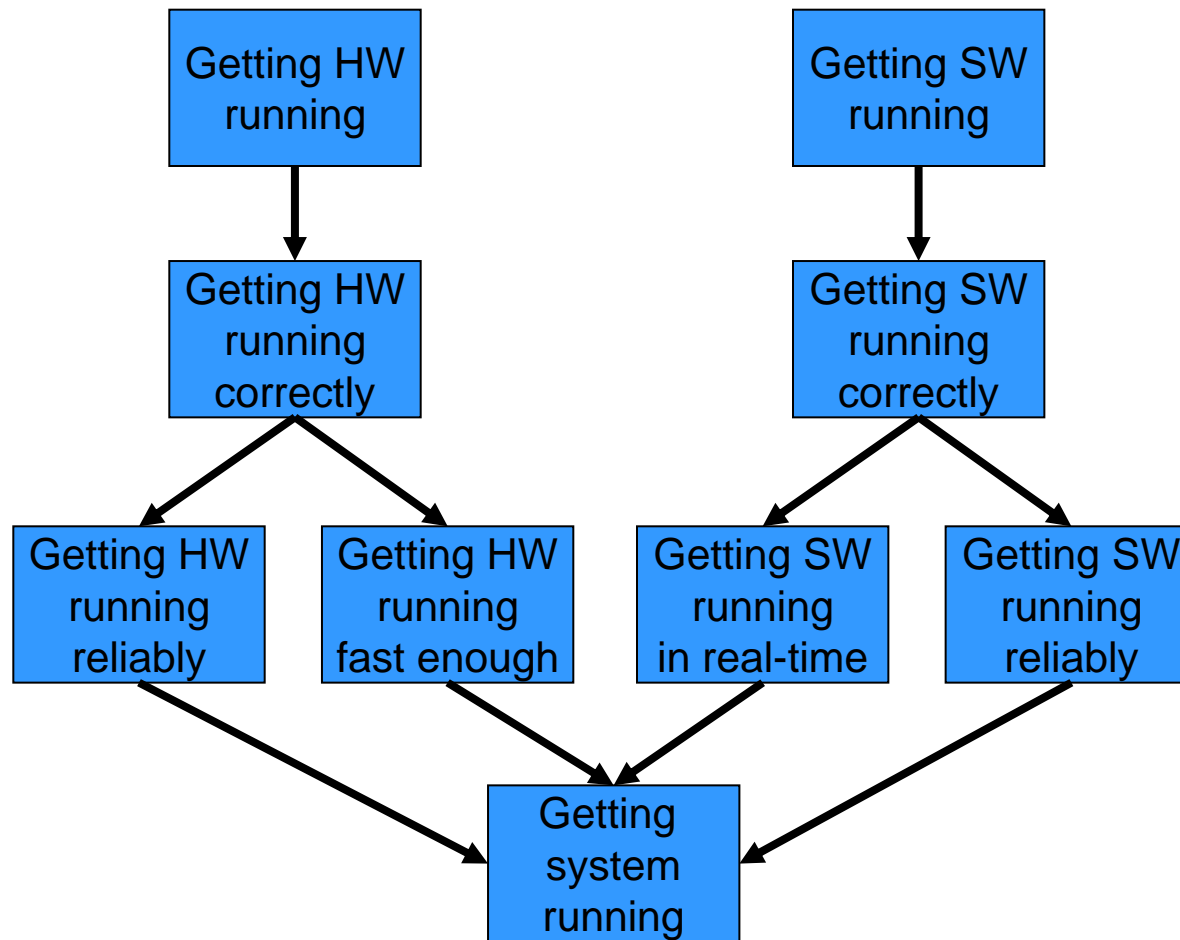
CpE-450 Spring 06

Class 6

Bruce McNair

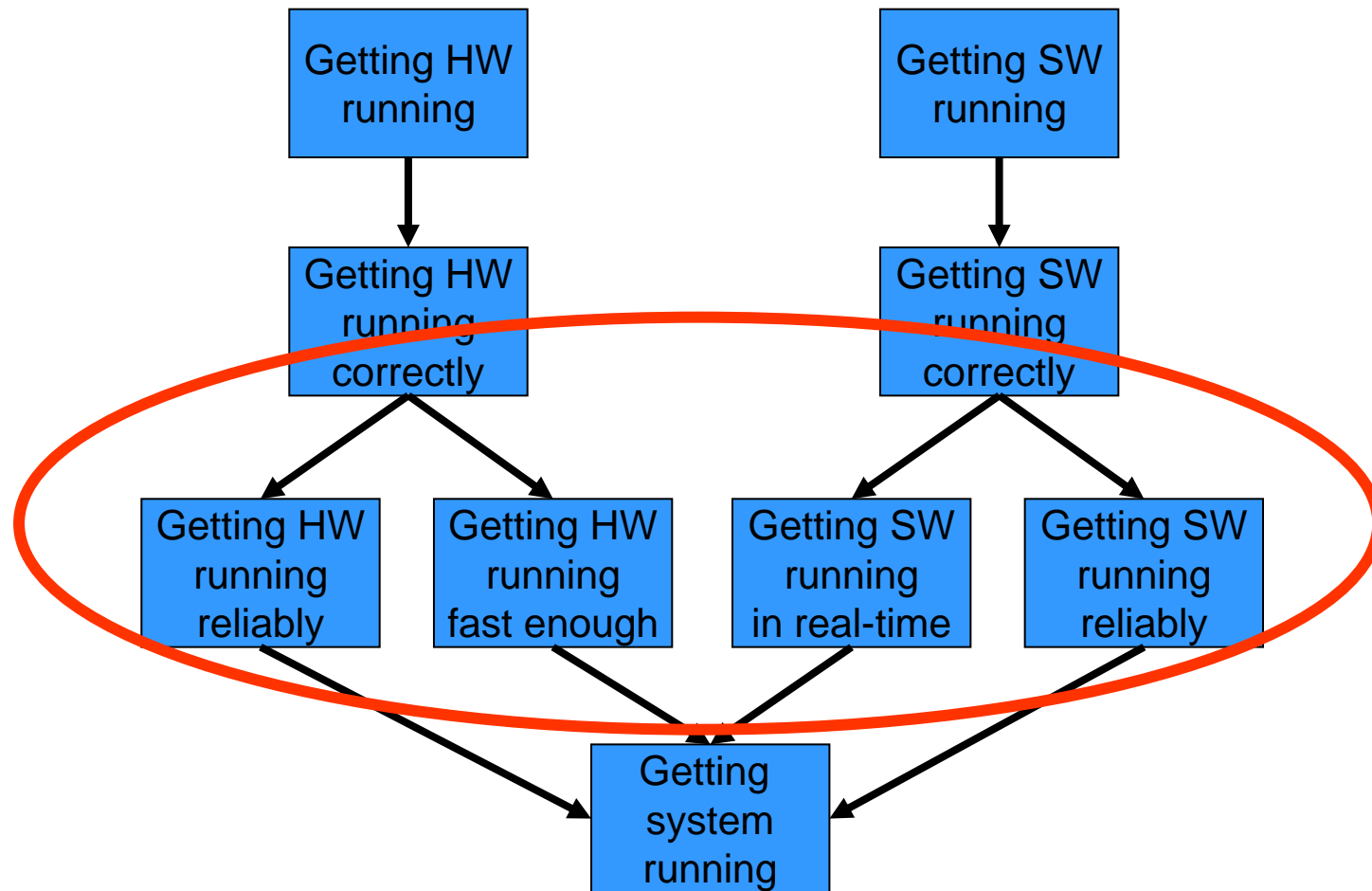
bmcnair@stevens.edu

Getting a Real-Time Embedded System Running



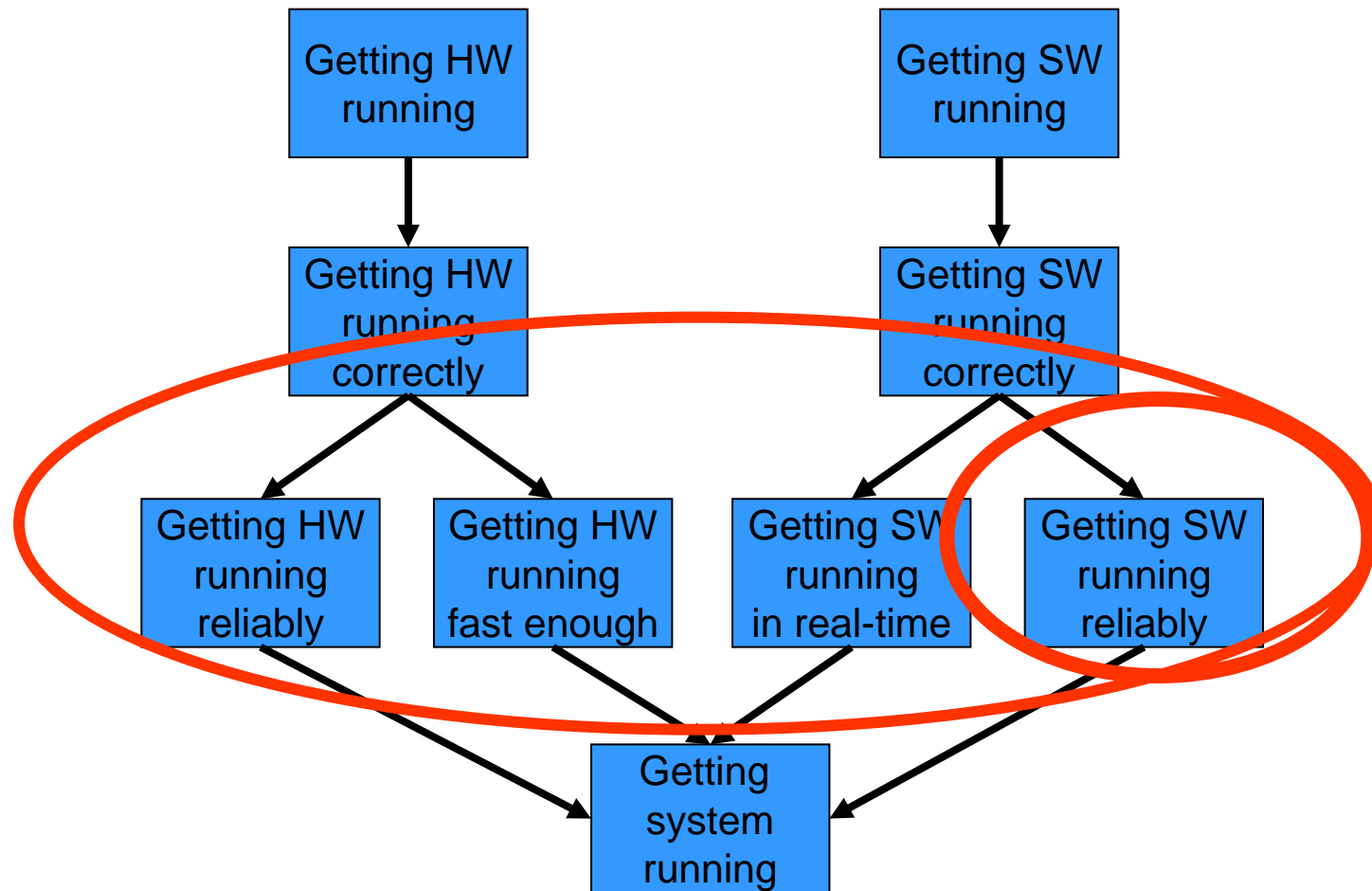
Getting a Real-Time Embedded System Running

- Evaluating performance and correctness

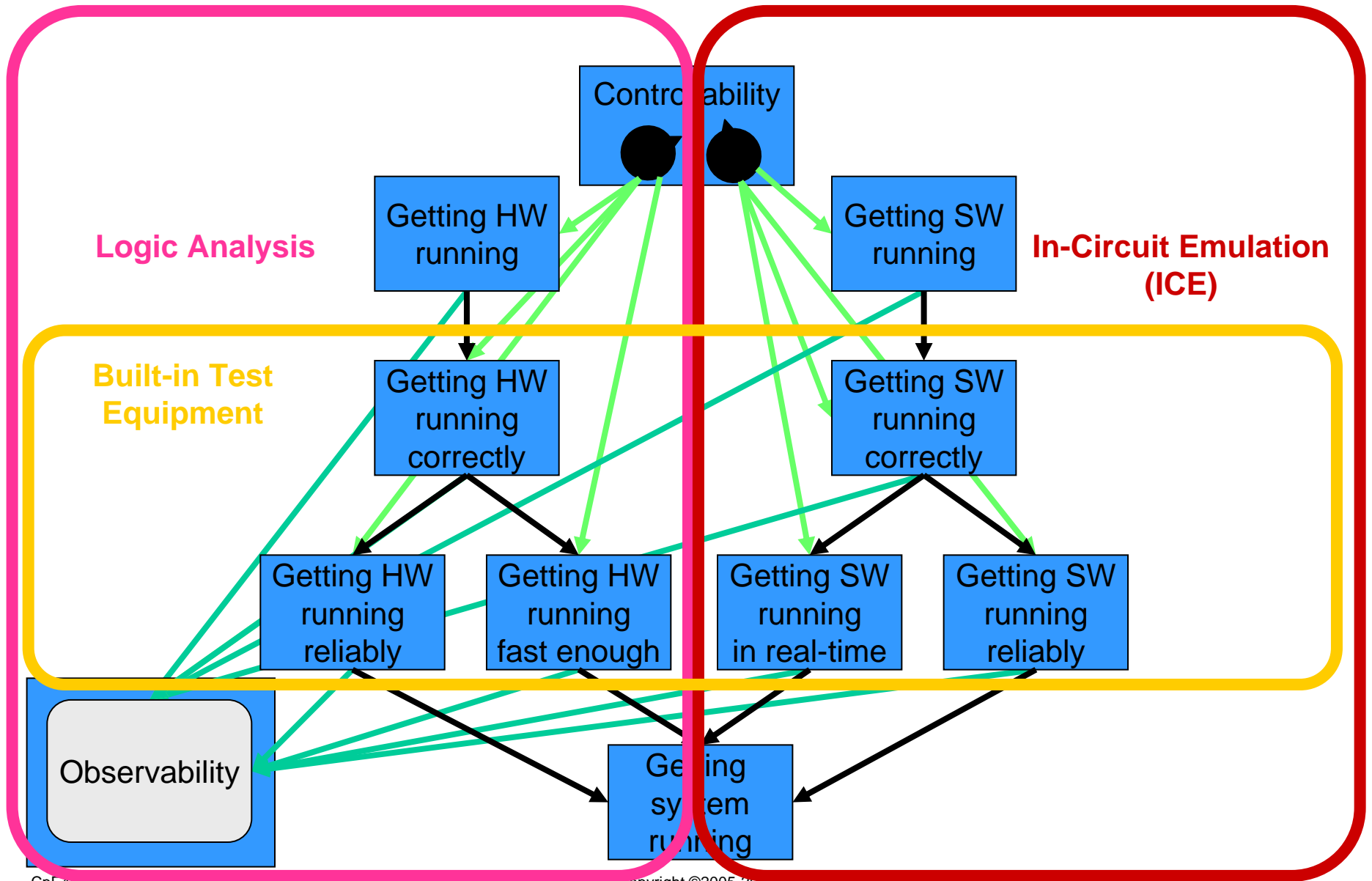


Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



Getting a Real-Time Embedded System Running



Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor (sometimes):

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i;  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i; ←  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

What is the initial value of i?

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i; ←  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

What is the initial
value of i?

PC's OS probably initialized
memory to 0
Embedded system does whatever
you programmed

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

Initial Conditions

- This code will probably work on your PC:

```
void main()  
{  
  int i; ←  
  float sine(100);  
  while (i++ != 100)  
  {  
    sine(i) = sin(i*2*pi/100)  
  }  
  do_something_with_sine_table(sine);  
}
```

What is the initial
value of i?

PC's OS probably initialized
memory to 0

Embedded system does whatever
you programmed

Next execution of program picks up
where previous execution left off.

- Results on an embedded processor:
 - sine table size is 100
 - sine table size is <100
 - sine table size is >100
 - program runs forever
 - program crashed unexpectedly
 - first, second and Nth run of program give different results

System Initialization

- Embedded programmer must initialize:
 - All variables that are declared
 - All peripheral controls
 - Stack location
 - Initial stack pointer
 - .
 - .
 - .

Memory Leakage

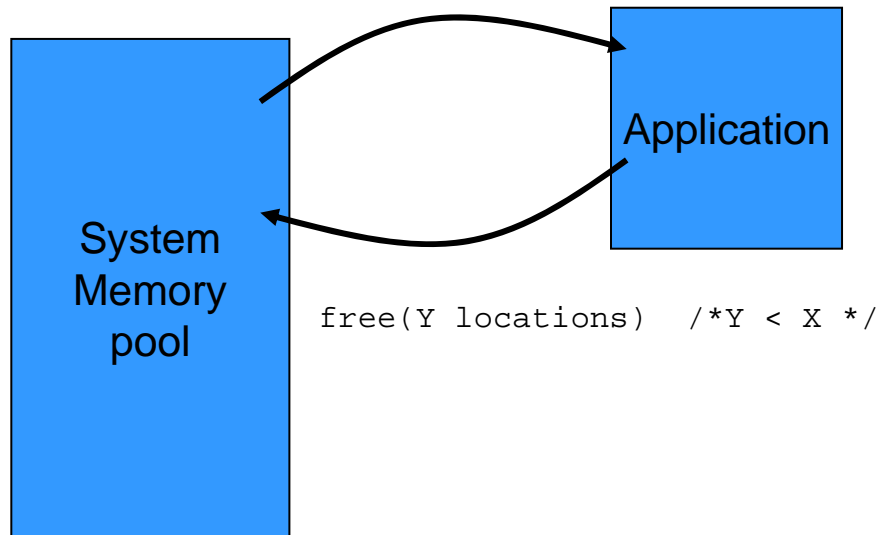
- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.

Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?

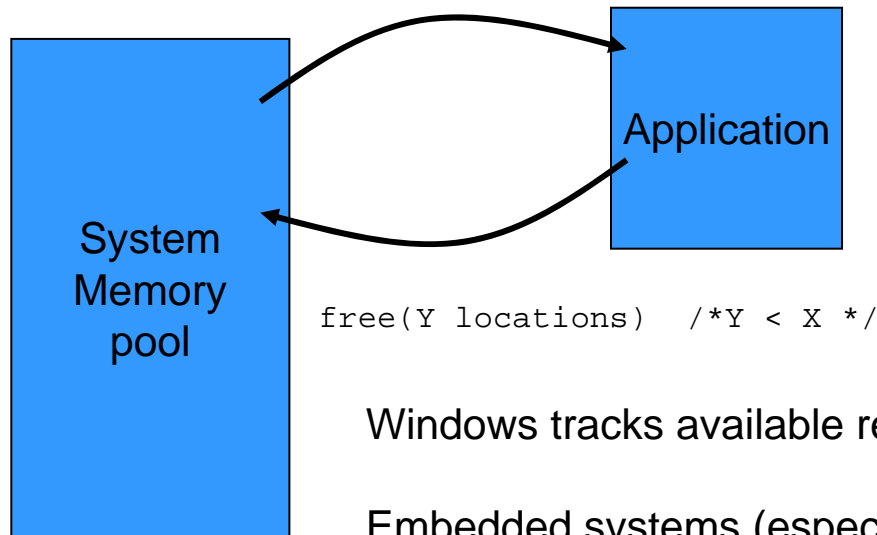
Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?



Memory Leakage

- Try this at home:
 1. Open N instances of your favorite application (Netscape, Explorer, Matlab, Word, etc.)
 2. Close them all
 3. Repeat until Windows complains about being short of memory, handles, or other resource.
- What is going wrong?



Windows tracks available resources and still gets into trouble

Embedded systems (especially real-time systems)
don't track resources

Embedded System Constraints

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
  if(i == 1)
  {
    return(1);
  }
  else
  {
    return(i*factorial(i-1));
  }
}
```

Embedded System Constraints

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
  if(i == 1)
  {
    return(1);
  }
  else
  {
    return(i*factorial(i-1));
  }
}
```

- What does this require from system?

Embedded System Constraints – Stack Overflow

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
    if(i == 1)
    {
        return(1);
    }
    else
    {
        return(i*factorial(i-1));
    }
}
```

- What does this require from system?
 - At each call of factorial(), return address and machine state are pushed on stack
 - There is no protected memory, so stack grows without restriction

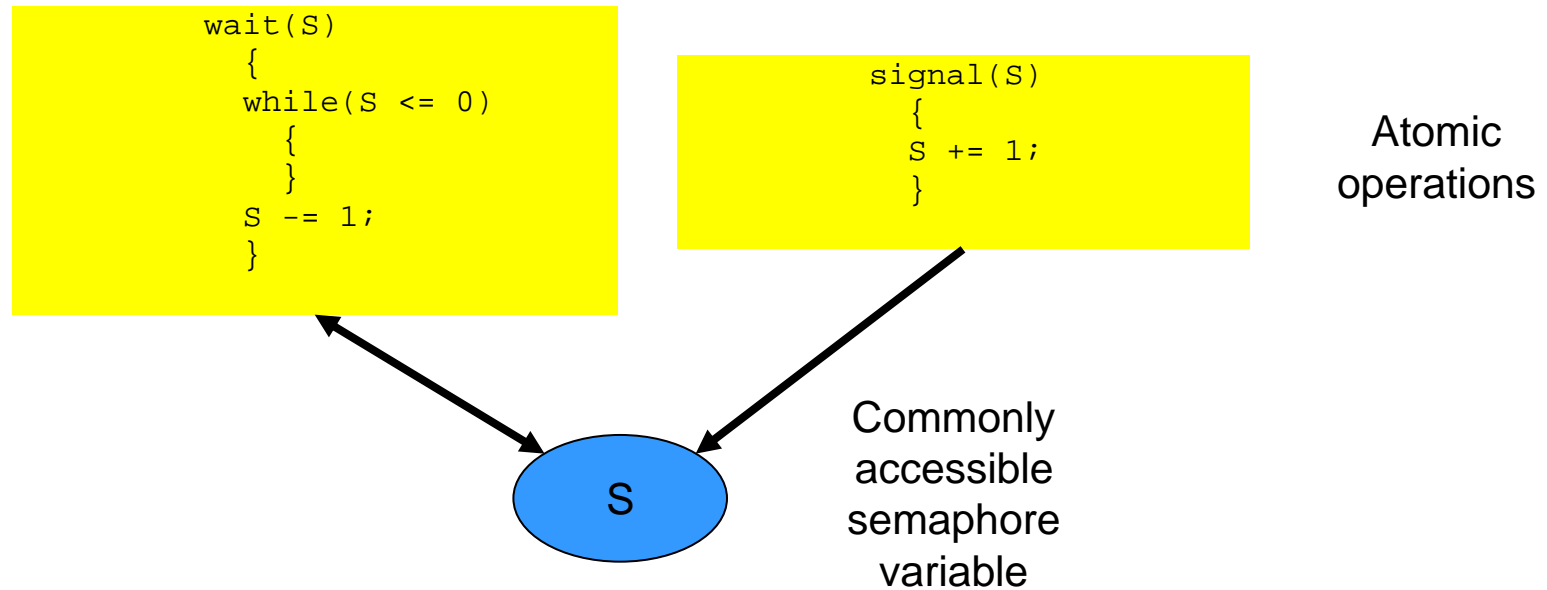
Embedded System Constraints – Stack Overflow

- Recursion: a simple way to write iterative code

```
long int factorial(int i)
{
    if(i == 1)
    {
        return(1);
    }
    else
    {
        return(i*factorial(i-1));
    }
}
```

- What does this require from system?
 - At each call of factorial(), return address and machine state are pushed on stack
 - There is no protected memory, so stack grows without restriction
 - Until it collides with data, program code, or peripheral

Semaphores for Interprocess Communications



Volatility

- Consider this code:

```
void main()
{
  int semaphore_S;
  do_something();
  wait(&semaphore_S);
  do_something_else();
}
```

```
void wait(int *S)          /* pass the semaphore's address to wait() */
{
  while(*S <= 0)
  {
  }
  *S -= 1;
}
```

```
signal(S)
{
  S += 1;
}
```

Other process

Volatility

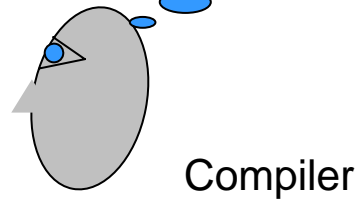
- Consider this code:

```
void main()  
{  
  int semaphore_S;  
  do_something();  
  wait(&semaphore_S);  
  do_something_else();  
}
```

```
void wait(int *S) /* pass address to wait() */  
{  
  while(*S <= 0)  
  {  
  }  
  *S -= 1;  
}
```

```
signal(S)  
{  
  S += 1;  
}
```

S is not modified in while() loop. I can improve this!



Volatility

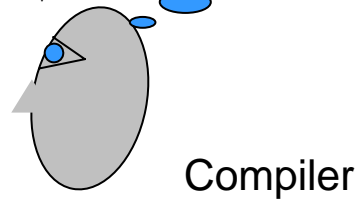
- Consider this code:

```
void main()
{
    int semaphore_S;
    do_something();
    wait(&semaphore_S);
    do_something_else();
}
```

```
signal(S)
{
    S += 1;
}
```

```
void wait(int *S) /* pass address to wait() */
{
    /* while(*S <= 0) */
    if(*S <= 0)
    {
        halt();
    }
    *S -= 1;
}
```

S is not modified in while() loop. I can improve this!



Volatility

- Consider this code:

```
void main()
{
  int semaphore_S;
  do_something();
  wait(&semaphore_S);
  do_something_else();
}
```

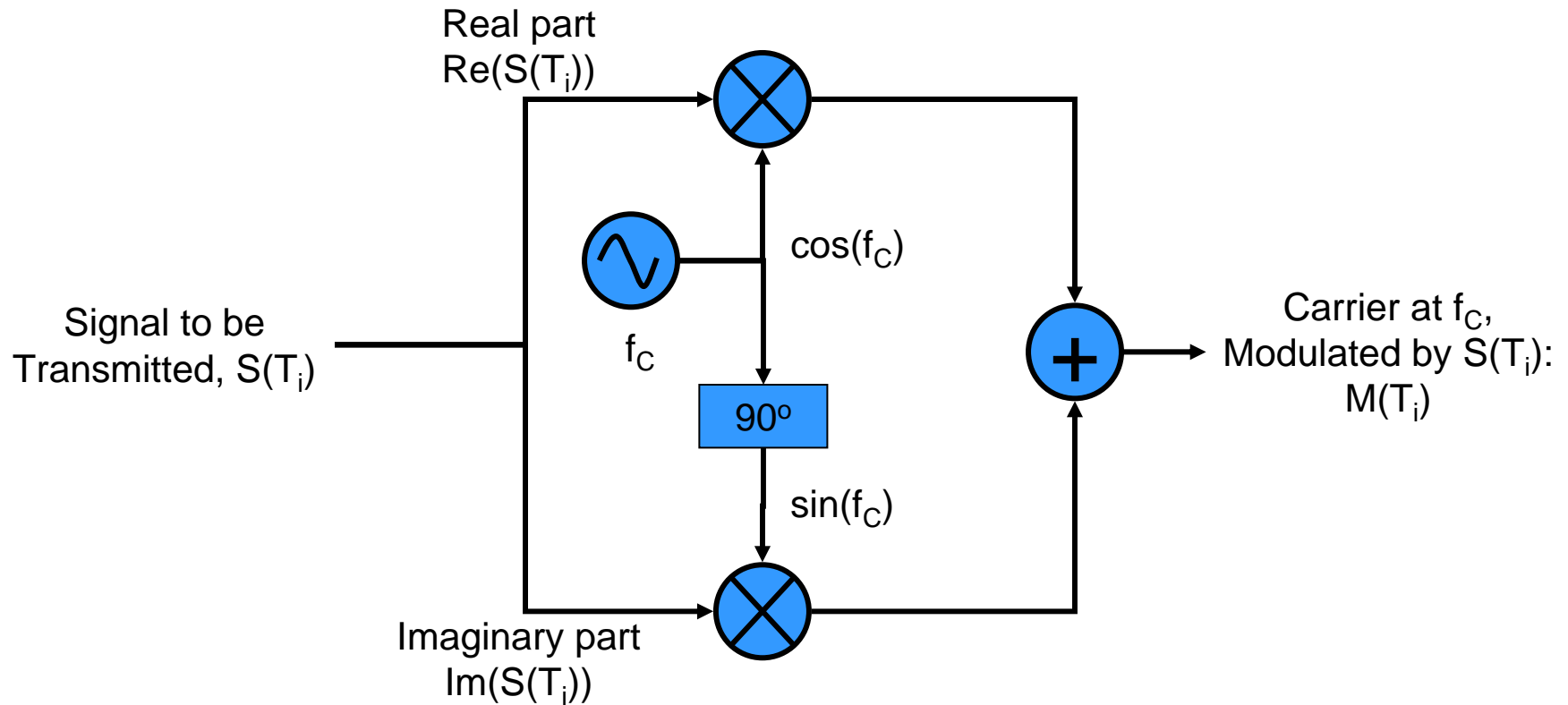
```
signal(S)
{
  S += 1;
}
```

```
void wait(volatile int *S)      /* pass the semaphore's address to wait() */
{
  while(*S <= 0)
  {
  }
  *S -= 1;
}
```

Variable must be considered to be asynchronously modified

Consider This Function

- Typical operation performed in analog modems, cellular phones, other signal processing devices



$$M(T_i) = \text{Re}(S(T_i)) \cdot \cos(2 \cdot \pi \cdot T_i) + \text{Im}(S(T_i)) \cdot \sin(2 \cdot \pi \cdot T_i)$$

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,  
              float fc, deltaT)  
{  
}
```

```
/* assume a complex signal  
   type is defined */  
struct signal  
{  
    float real;  
    float imag;  
};
```

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,  
              float fc, deltaT)  
{  
}
```

```
/* assume a complex signal  
   type is defined */  
struct signal  
{  
    float real;  
    float imag;  
};
```

```
/* modulate is called with  
   arrays (of the structure)  
   representing the input and  
   output signals. Assume there  
   are N samples */  
signal baseband[N];  
float modulated[N];  
...  
baseband[i] = value;...  
...  
/* assume that the carrier  
   frequency and sample period are  
   given */  
modulate(*baseband, *modulated, N,  
         fc, deltaT);
```

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    float Ti=0;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(2*pi*Ti*fc)+
              S[i].imag*sin(2*pi*Ti*fc);
        Ti += deltaT;
    }
}
```

```
/* assume a complex signal
   type is defined */
struct signal
{
    float real;
    float imag;
};
```

```
/* modulate is called with
   arrays (of the structure)
   representing the input and
   output signals. Assume there
   are N samples */
signal baseband[N];
float modulated[N];
...
baseband[i] = value;...
...
/* assume that the carrier
   frequency and sample period are
   given */
modulate(*baseband, *modulated, N,
        fc, deltaT);
```

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    float Ti=0;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(2*pi*Ti*fc)+
              S[i].imag*sin(2*pi*Ti*fc);
        Ti += deltaT;
    }
}
```

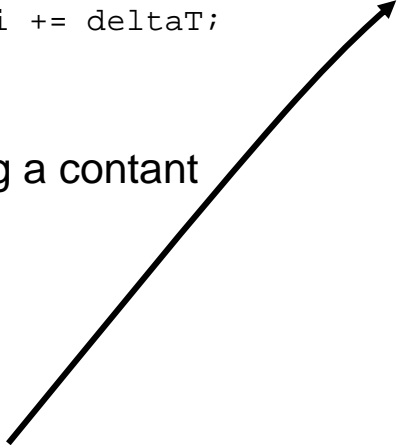
Real-time performance issues:

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    float Ti=0;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(2*pi*Ti*fc)+
              S[i].imag*sin(2*pi*Ti*fc);
        Ti += deltaT;
    }
}
```

Recalculating a constant
(twice)



Real-time performance issues:

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
  int i;
  float Ti=0;
  for(i=0; i<N, i++)
  {
    M[i] = S[i].real*cos(2*pi*Ti*fc)+
          S[i].imag*sin(2*pi*Ti*fc);
    Ti += deltaT;
  }
}
```

Calculating trig functions



Real-time performance issues:

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    float Ti=0;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(2*pi*Ti*fc)+
              S[i].imag*sin(2*pi*Ti*fc);
        Ti += deltaT;
    }
}
```

Correctness issues:

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,  
              float fc, deltaT)  
{  
    int i;  
    float Ti=0;  
    for(i=0; i<N, i++)  
    {  
        M[i] = S[i].real*cos(2*pi*Ti*fc)+  
              S[i].imag*sin(2*pi*Ti*fc);  
        Ti += deltaT;  
    }  
}
```

Each time modulate() is called, carrier phase is reset to zero

Correctness issues:

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float Ti=0;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(2*pi*Ti*fc)+
              S[i].imag*sin(2*pi*Ti*fc);
        Ti += deltaT;
    }
}
```

Ti is set to zero on first call of modulate().
Whatever value is left in the variable after
modulate() exits, remains.

Real-time code

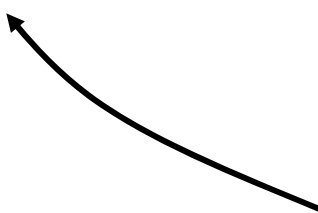
- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float Ti=0;
    #define twopi=6.28318531
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(twopi*Ti*fc)+
              S[i].imag*sin(twopi*Ti*fc);
        Ti += deltaT;
    }
}
```

Replace the calculation of 2π by a #define. #define's are computed at compile time and, while they appear to be variables, they are actually constants in the code:

```
M[i] = S[i].real*cos(6.28318531 *Ti*fc)+
       S[i].imag*sin(6.28318531 *Ti*fc);
```

$twopi*Ti*fc$ is computed repeatedly
Move that out of the loop.



Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
#define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(phi)+
              S[i].imag*sin(phi);
        phi += deltaPHI;
    }
}
```

$2\pi f_c T$ is computed repeatedly
Move that out of the loop.

Real-time code

- Example code for a real-time embedded system:

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
#define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    for(i=0; i<N, i++)
    {
        M[i] = S[i].real*cos(phi)+
              S[i].imag*sin(phi);
        phi += deltaPHI;
    }
}
```

This helps, but we still have 2 trig computations in the loop.

Real-time code

- Example code for a real-time embedded system:

```
float sine_table[SIZE], cosine_table[SIZE];

void initialize_sine_table(*sine_table, *cosine_table)
{
    int i;
    for(i=0;i<SIZE,i++)
    {
        sine_table[i] = sin(2*pi*i/SIZE);
        cosine_table[i] = cos(2*pi*i/SIZE);
    }
}
```

Trade memory for
computation

```
void modulate (signal *S, float *M, int N,
               float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi += deltaPHI;
    }
}
```

Real-time code

- Example code for a real-time embedded system:

```
float sine_table[SIZE], cosine_table[SIZE];

void initialize_sine_table(*sine_table, *cosine_table)
{
    int i;
    for(i=0;i<SIZE,i++)
    {
        sine_table[i] = sin(2*pi*i/SIZE);
        cosine_table[i] = cos(2*pi*i/SIZE);
    }
}
```

Trade memory for
computation

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi += deltaPHI;
    }
}
```

What is wrong with this code?

How can memory and speed
be improved?

Real-time code

- Example code for a real-time embedded system:

```
float sine_table[SIZE], cosine_table[SIZE];

void initialize_sine_table(*sine_table, *cosine_table)
{
    int i;
    for(i=0;i<SIZE,i++)
    {
        sine_table[i] = sin(2*pi*i/SIZE);
        cosine_table[i] = cos(2*pi*i/SIZE);
    }
}
```

Trade memory for computation

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi += deltaPHI;
    }
}
```

What is wrong with this code?

“phase” can exceed “SIZE”

How can memory and speed be improved?

Real-time code

- Example code for a real-time embedded system:

```
float sine_table[SIZE], cosine_table[SIZE];

void initialize_sine_table(*sine_table, *cosine_table)
{
    int i;
    for(i=0;i<SIZE,i++)
    {
        sine_table[i] = sin(2*pi*i/SIZE);
        cosine_table[i] = cos(2*pi*i/SIZE);
    }
}
```

Trade memory for
computation

```
void modulate (signal *S, float *M, int N,
               float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase] +
               S[i].imag*sine_table[phase];
        phi += deltaPHI;
        if (phi>twopi)
        {
            phi -= twopi;
        }
    }
}
```

What is wrong with this code?

“phase” can exceed “SIZE”

How can memory and speed
be improved?

Real-time code

- Example code for a real-time embedded system:

```
float sine_table[SIZE], cosine_table[SIZE];

void initialize_sine_table(*sine_table, *cosine_table)
{
    int i;
    for(i=0;i<SIZE,i++)
    {
        sine_table[i] = sin(2*pi*i/SIZE);
        cosine_table[i] = cos(2*pi*i/SIZE);
    }
}
```

Trade memory for computation

```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
```

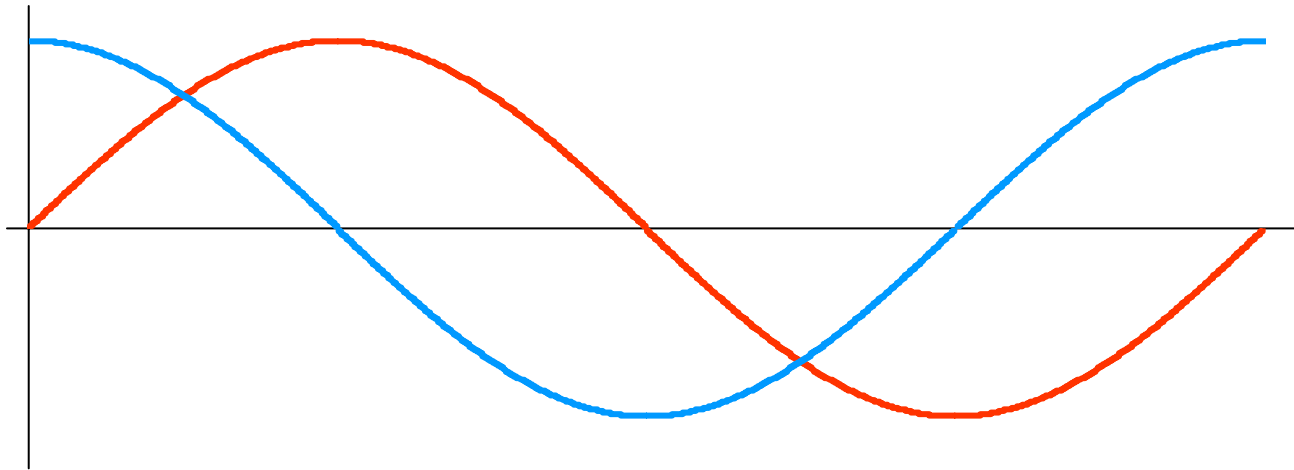
```
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi += deltaPHI;
        if (phi>twopi)
        {
            phi -= twopi;
        }
    }
}
```

What is wrong with this code?

How can memory and speed be improved?

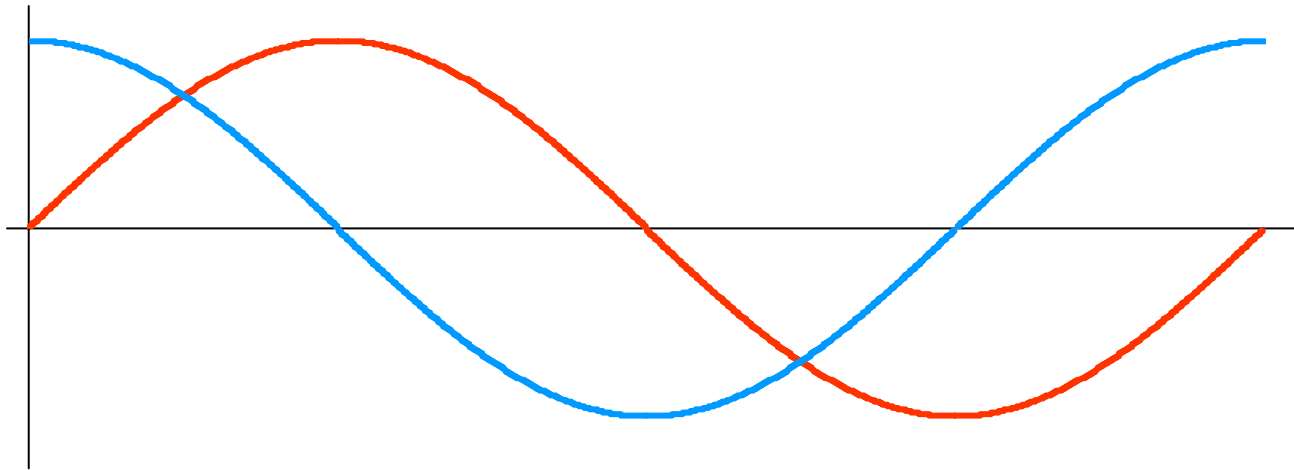
Minimizing Sine Table Memory

- Consider the characteristics of sine and cosine:

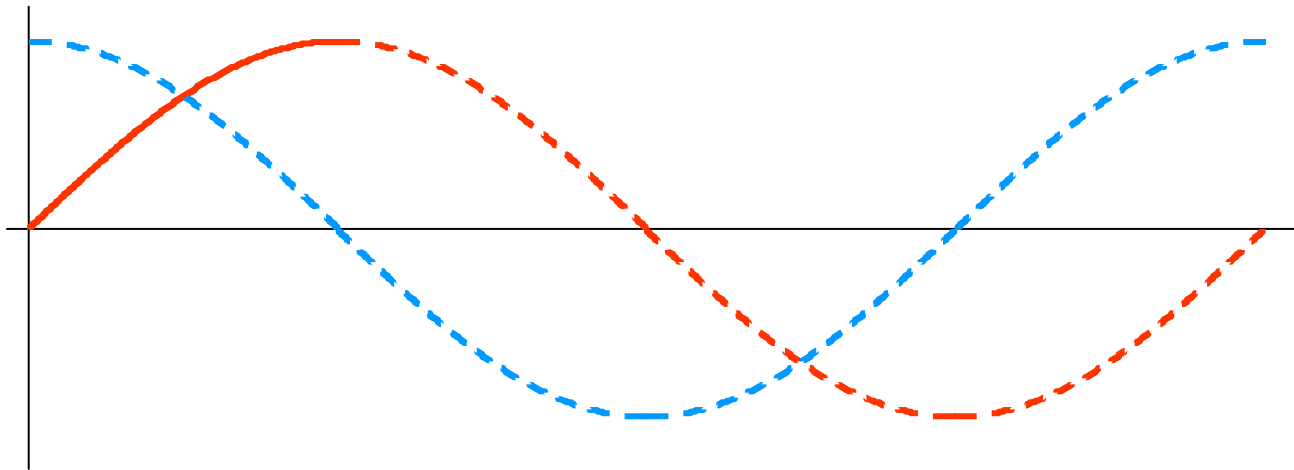


Minimizing Sine Table Memory

- Consider the characteristics of sine and cosine:



- One quarter of the sine can be used to recreate the rest of the data:



Assignment 5

- Modify the code presented in these slides to run on your PC. E.g., you will need a main() program to call the modulate routine
- Set up the program to allow you to time execution of the various forms of the modulate() routine. Compare the execution times.
- Implement any ways you can find to reduce memory requirements or improve execution time.