Real-Time Embedded Systems

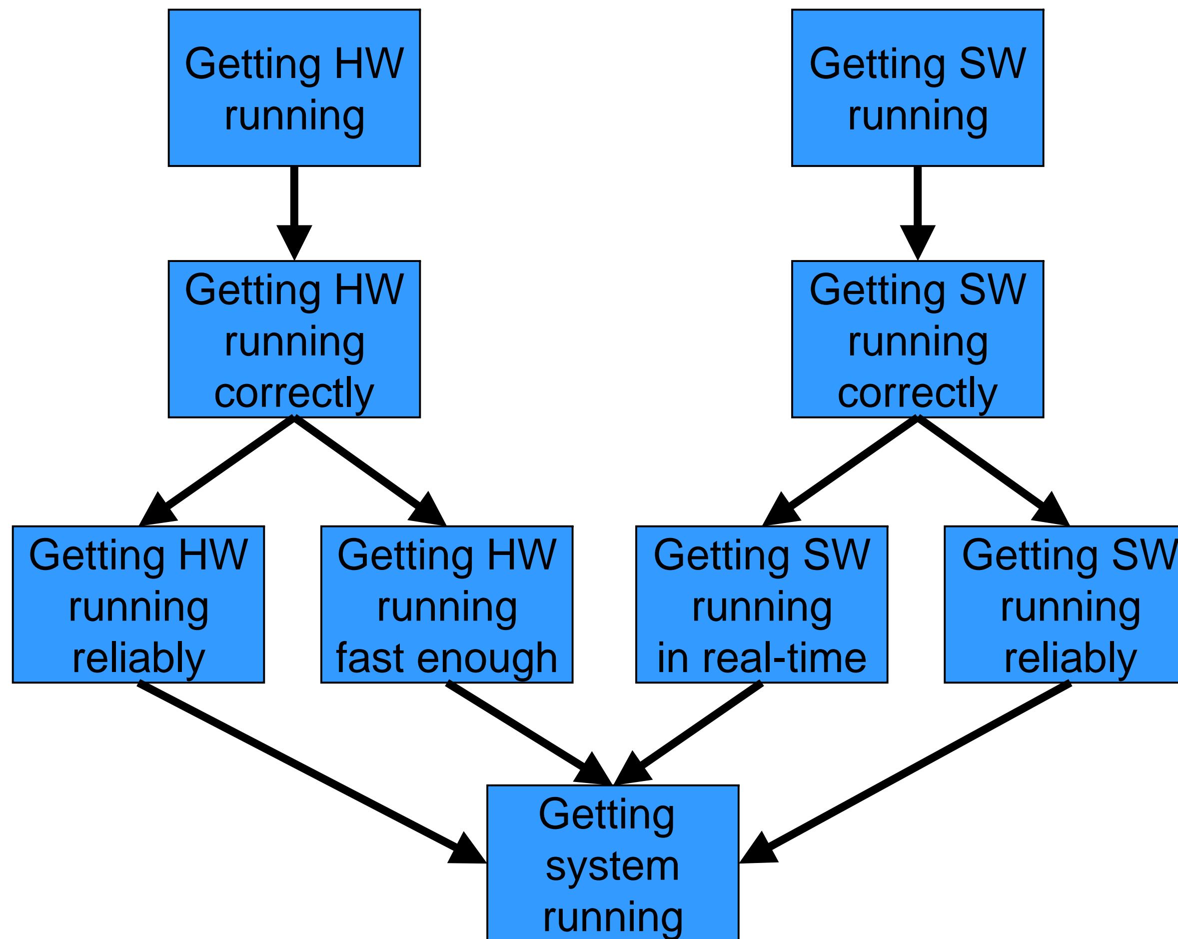
CpE-450 Spring 06

Class 7

Bruce McNair
bmcnair@stevens.edu

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

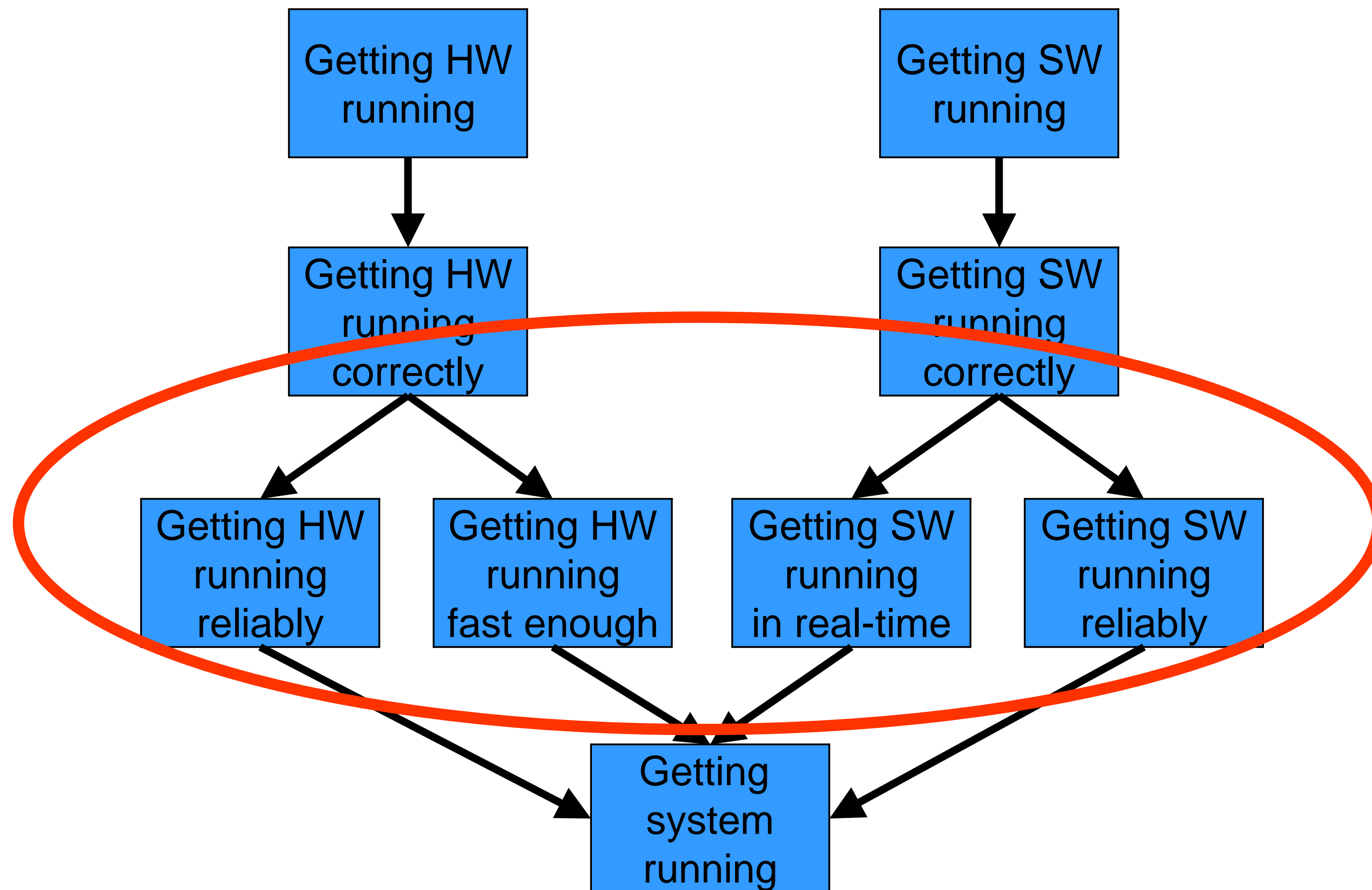
Getting a Real-Time Embedded System Running



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Getting a Real-Time Embedded System Running

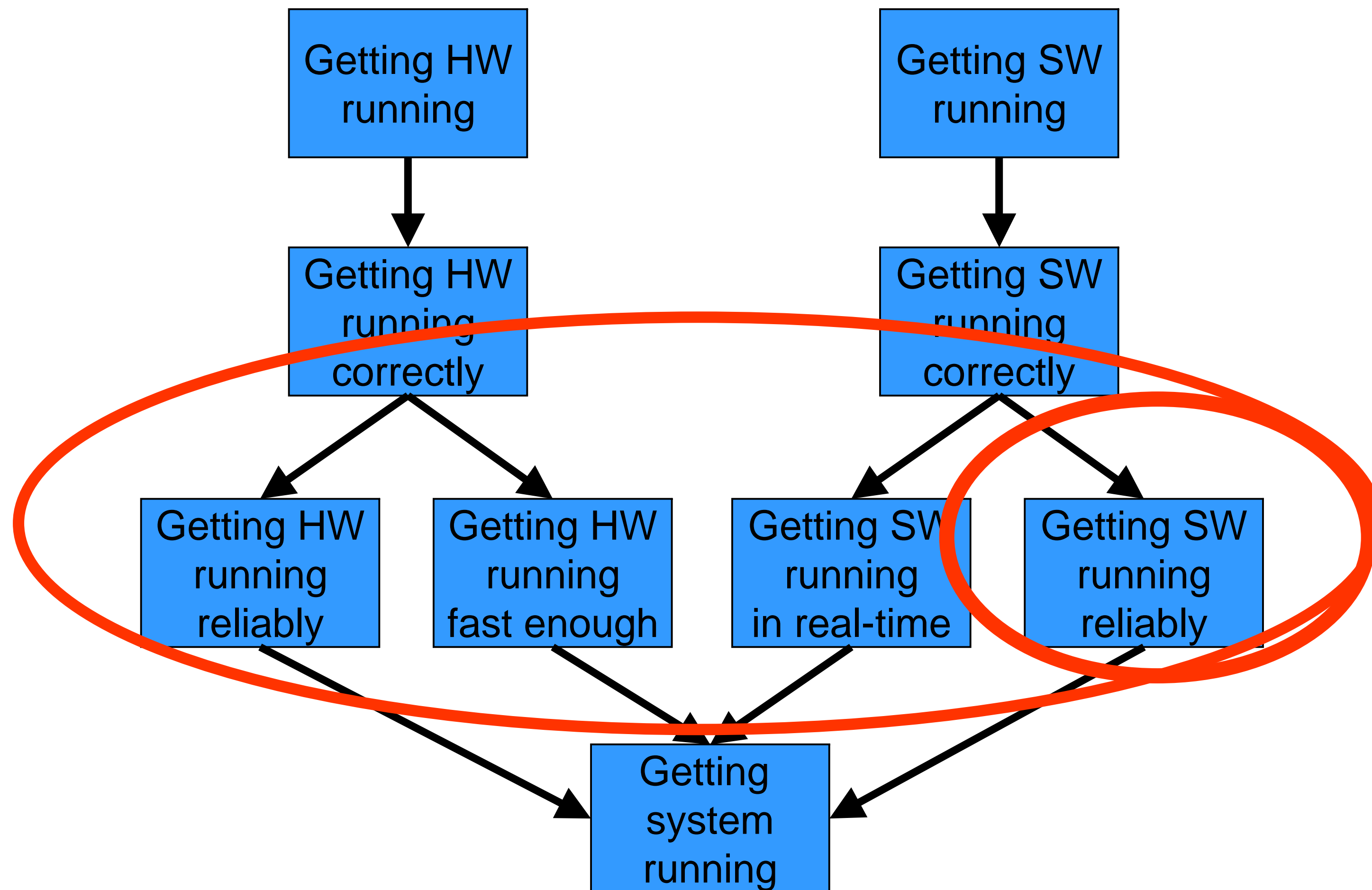
- Evaluating performance and correctness



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

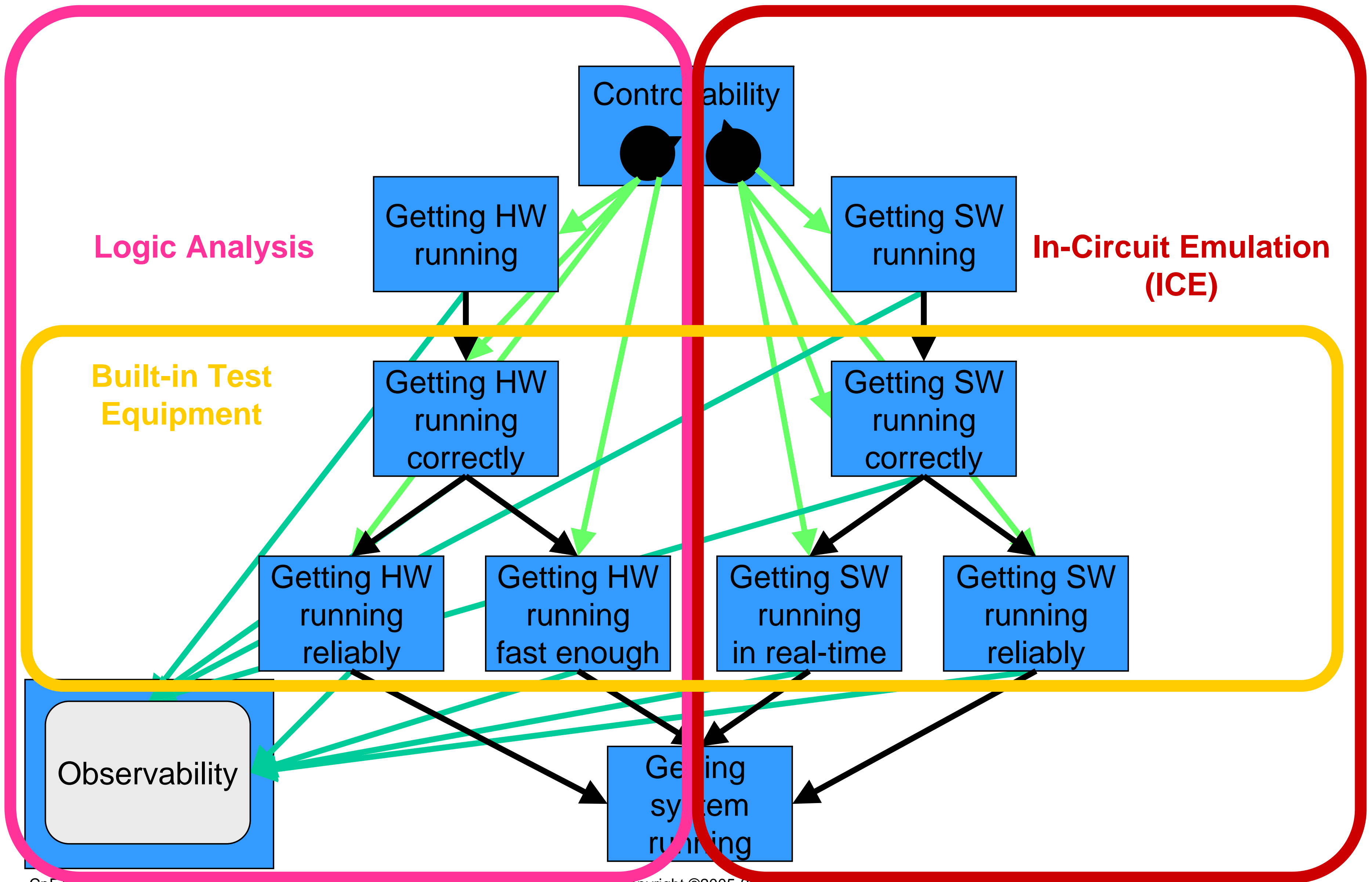
Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Getting a Real-Time Embedded System Running



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
  }
```


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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):

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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 N^{th} run of program give different results

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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 N[th] run of program give different results

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

System Initialization

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

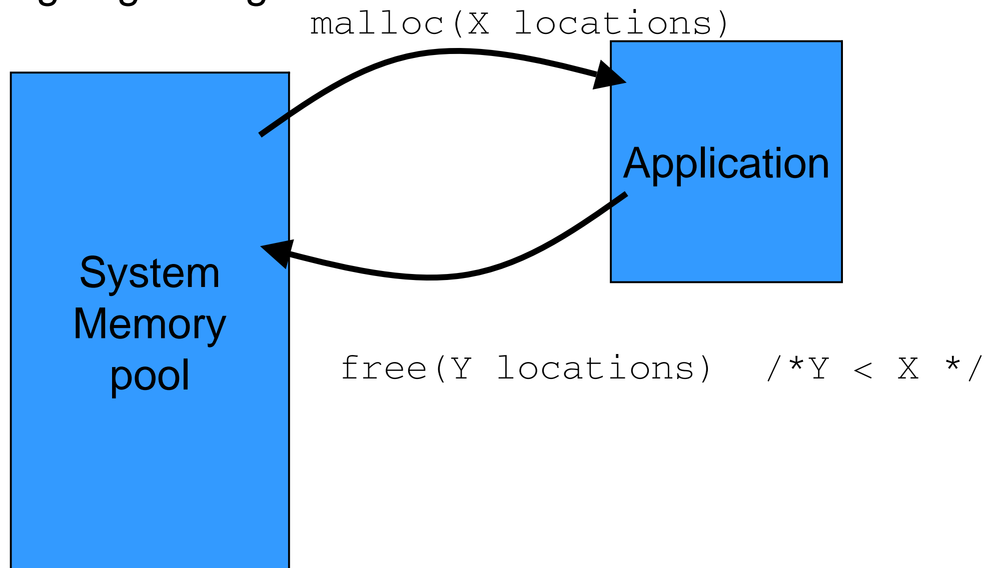
- What is going wrong?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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?

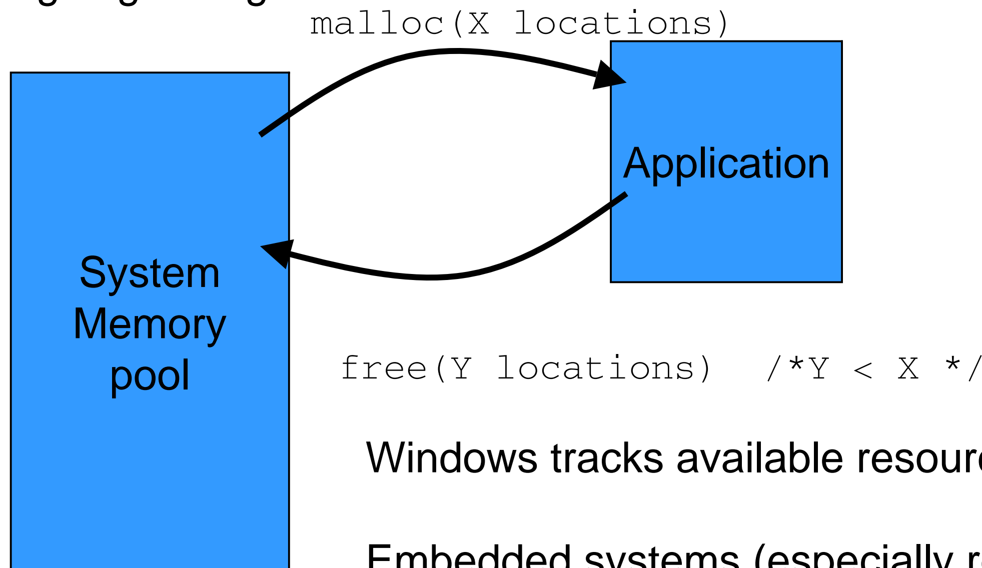


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
      }
  }
```


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Embedded System Constraints – Stack Overflow

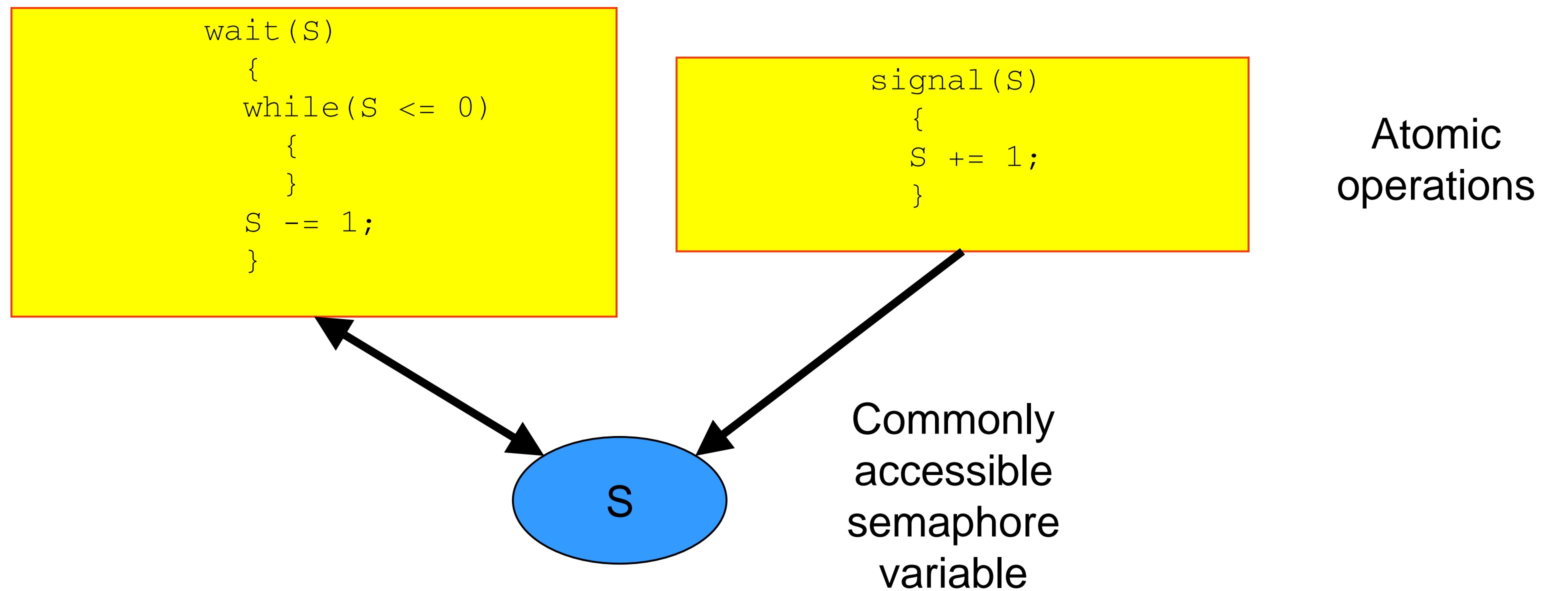
- Recursion: a simple way to write iterative code

```
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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Semaphores for Interprocess Communications



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Volatility

- Consider this code:

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

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

Other process

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

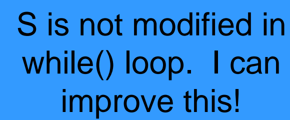

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Volatility

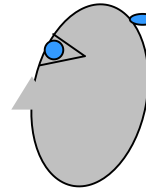
- Consider this code:

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

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




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



Compiler

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

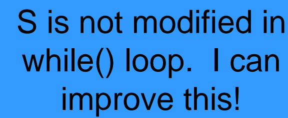
Volatility

- Consider this code:

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

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

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



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Volatility

- Consider this code:

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

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

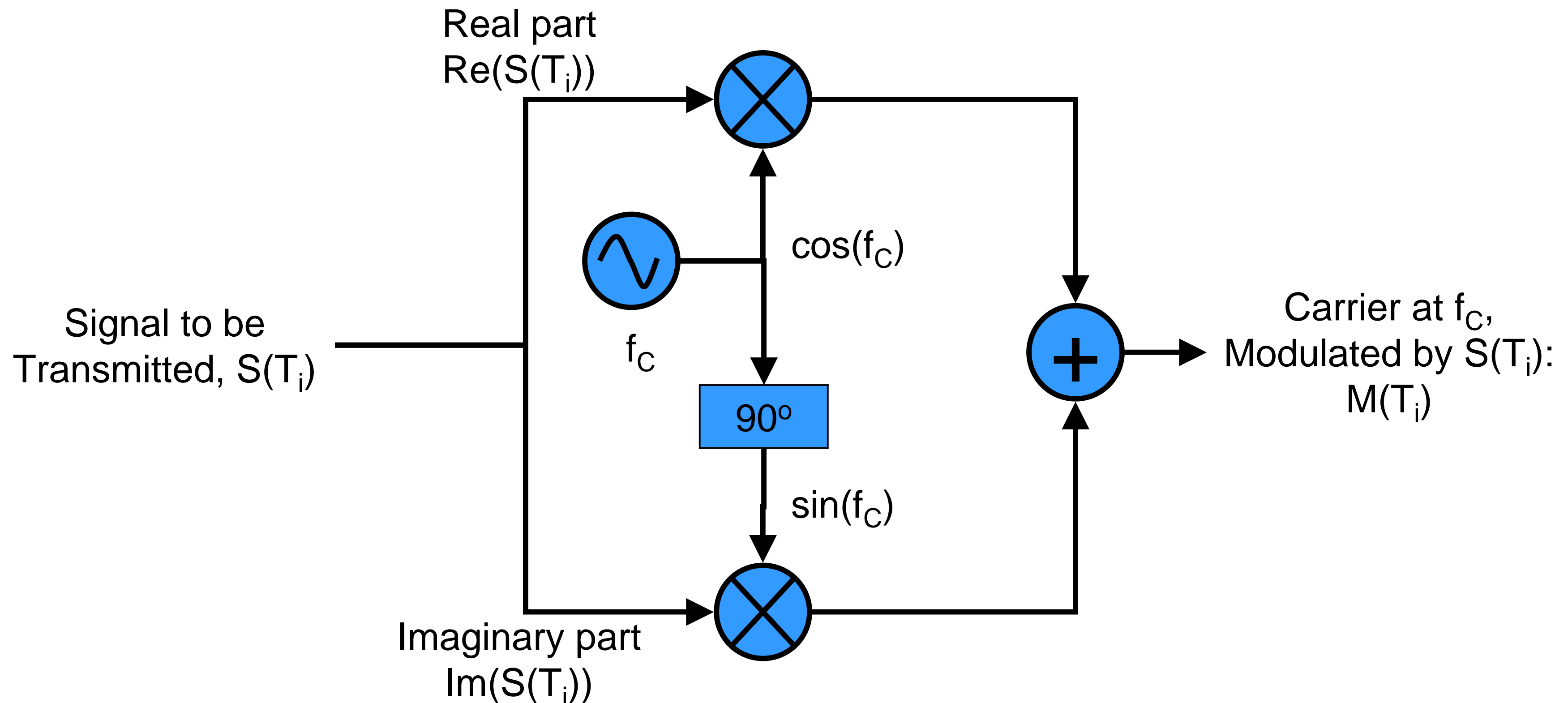
```
void wait(volatile int *S)        /* pass the semaphore's address to wait() */
  {
  while(*S <= 0)
    {
    }
  *S -= 1;
  }
```

Variable must be considered to be asynchronously modified

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Consider This Function

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



$$M(T_i) = \mathrm{Re}(S(T_i)) \bullet \cos(2 \bullet \pi \bullet T_i) + \mathrm{Im}(S(T_i)) \bullet \sin(2 \bullet \pi \bullet T)$$

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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;
  };
```


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
```


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
```


Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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 2*pi 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);
```

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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;
     }
   }
```

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

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
     }
   }

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;
     }
   }
```

Trade memory for computation

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
     }
   }

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;
     }
   }
```

Trade memory for computation

What is wrong with this code?

How can memory and speed be improved?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
     }
   }

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;
     }
   }
```

Trade memory for computation

What is wrong with this code?

“phase” can exceed “SIZE”

How can memory and speed be improved?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
     }
   }

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;
        }
     }
   }
```

Trade memory for computation

What is wrong with this code?

~~"phase" can exceed "SIZE"~~

How can memory and speed be improved?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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);
     }
   }

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;
        }
     }
   }
```

Trade memory for computation

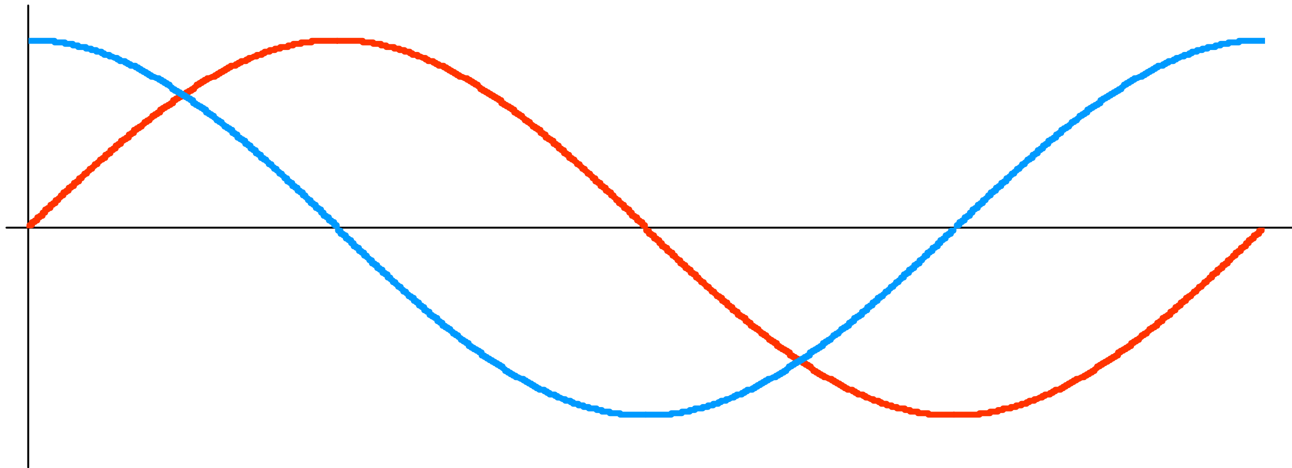
What is wrong with this code?

How can memory and speed be improved?

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

Minimizing Sine Table Memory

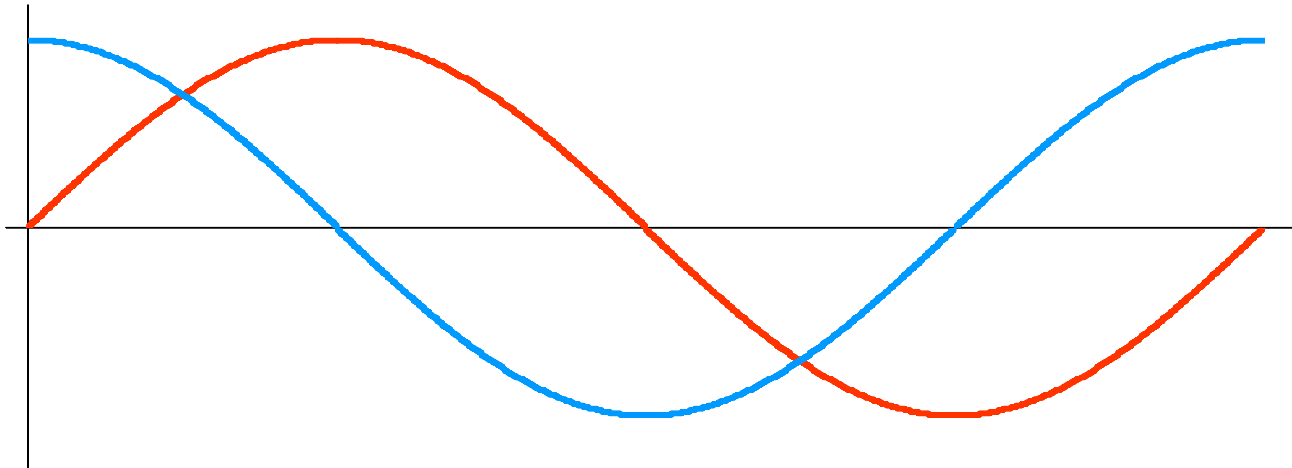
- Consider the characteristics of sine and cosine:



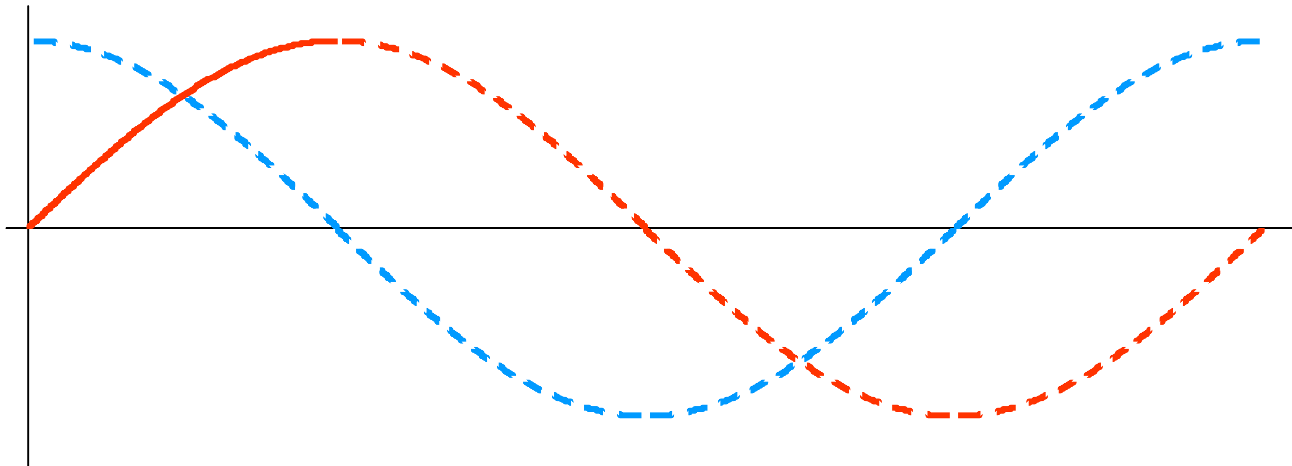
Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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:



Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

OPTIONAL

REQUIRED

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved

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.

OPTIONAL

- Enjoy Spring Break

REQUIRED

Copyright ©2005-2006
Stevens Institute of Technology - All rights reserved