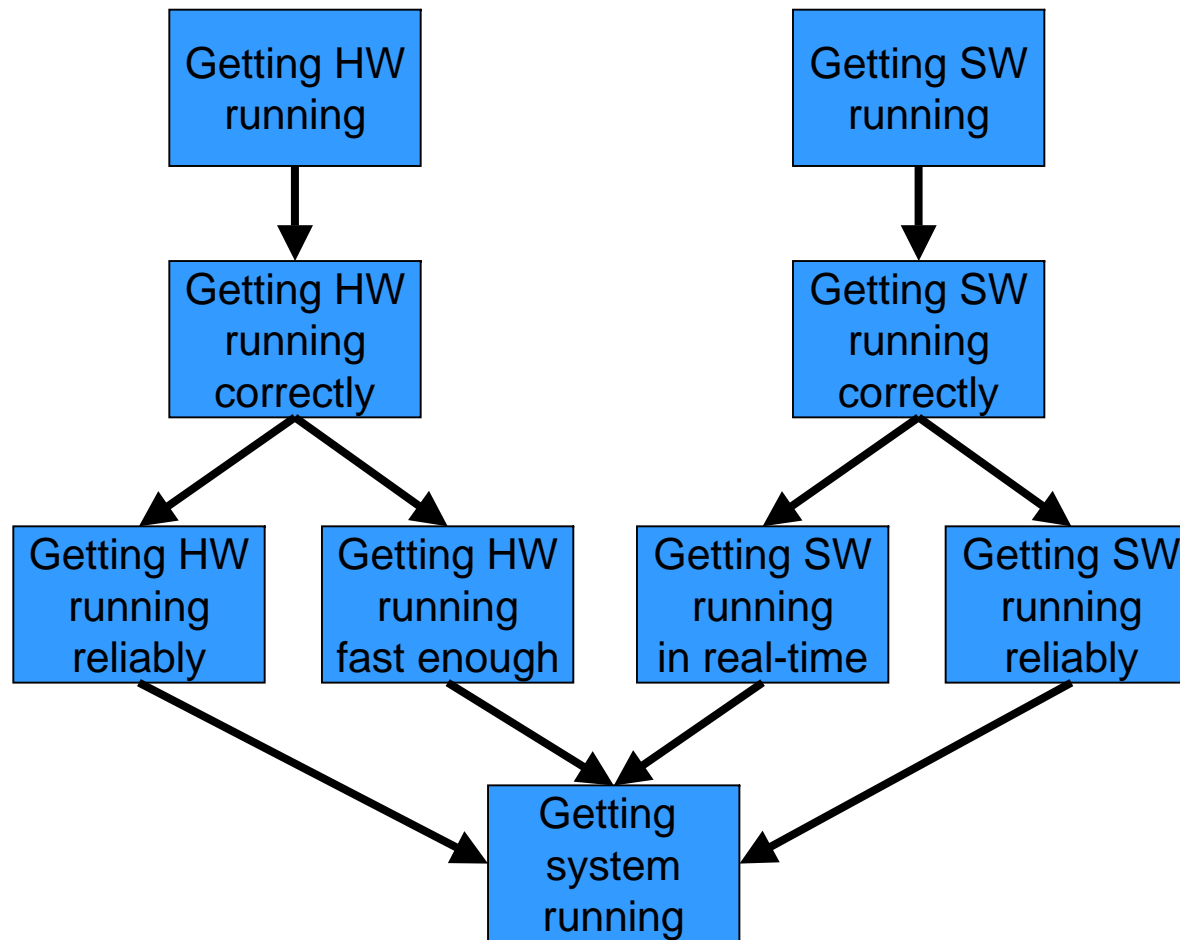


# Architecture, Design and Implementation of Embedded Systems for Real-Time Applications

CpE-450 - Spring 05  
Class 18

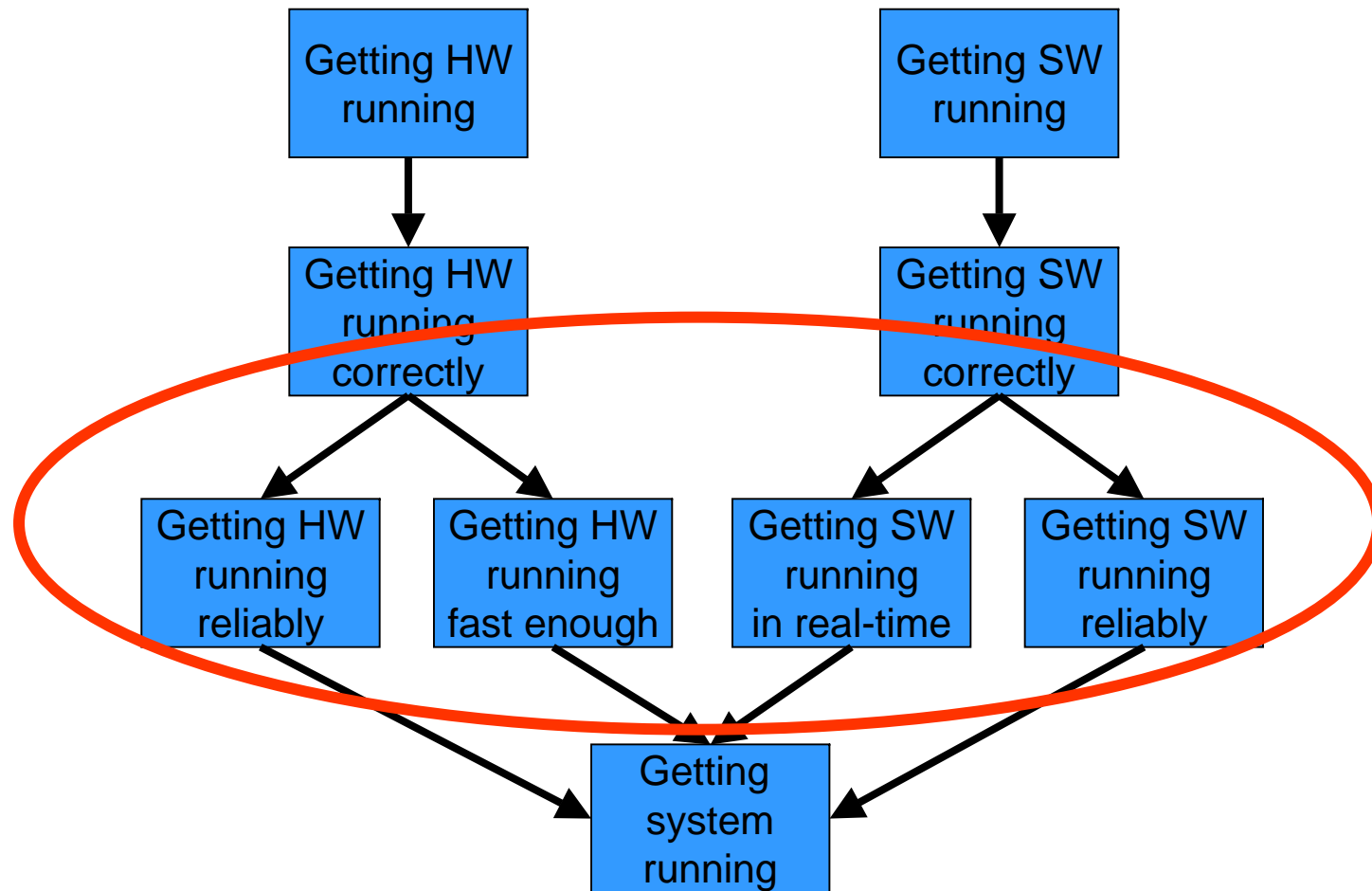
Bruce McNair  
bmcnair@stevens.edu

# Getting a Real-Time Embedded System Running



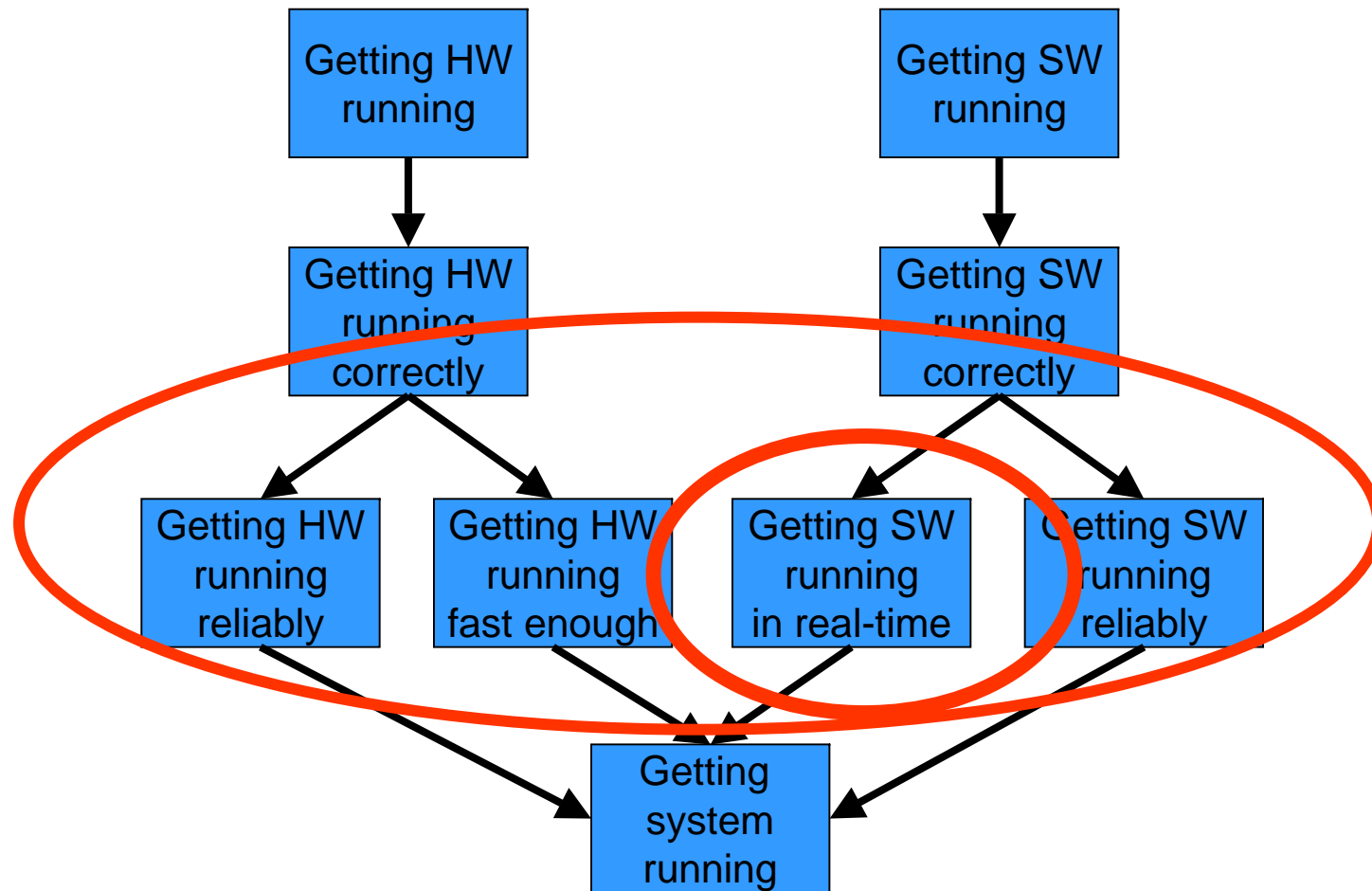
# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness

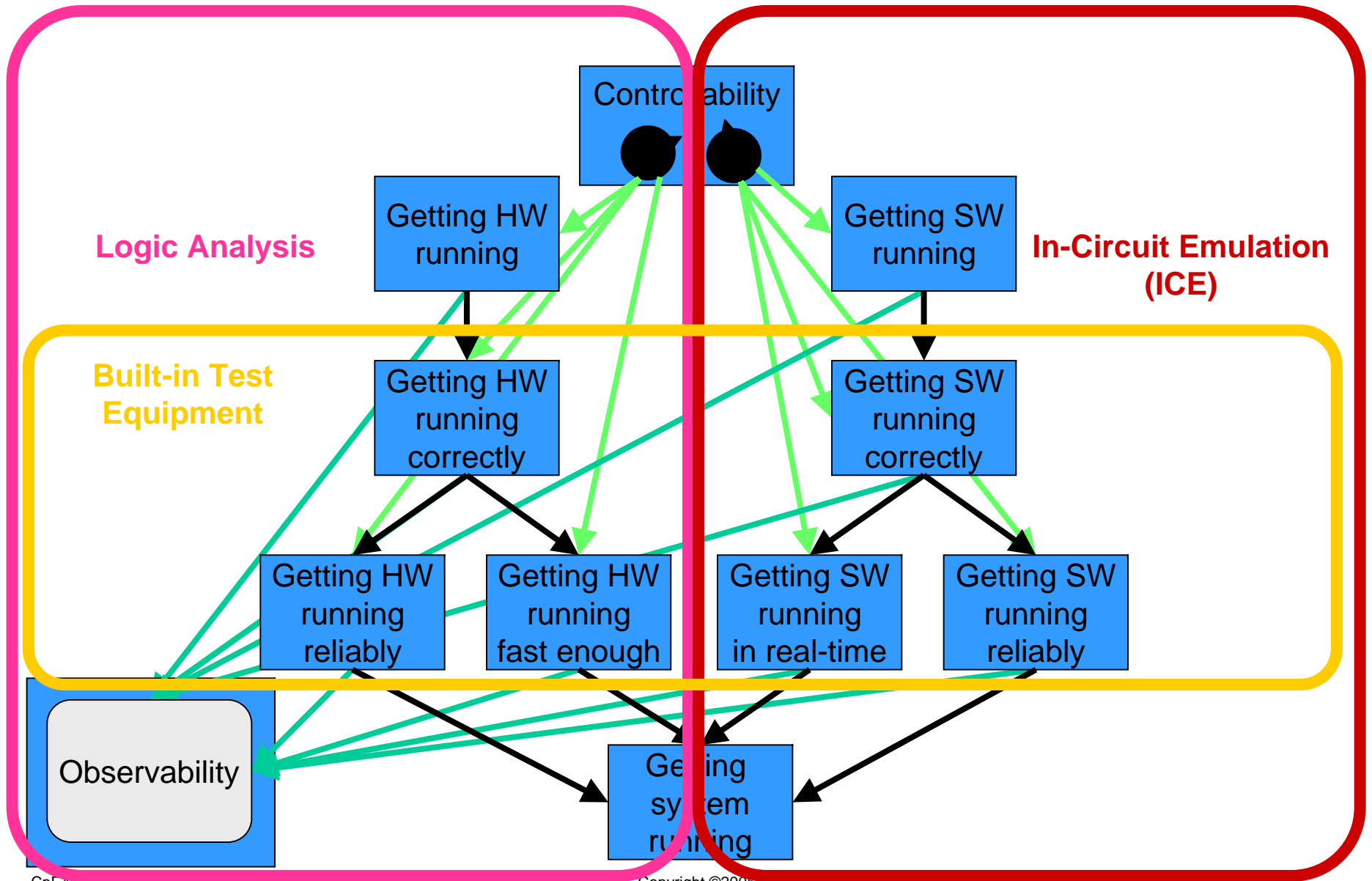


# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness

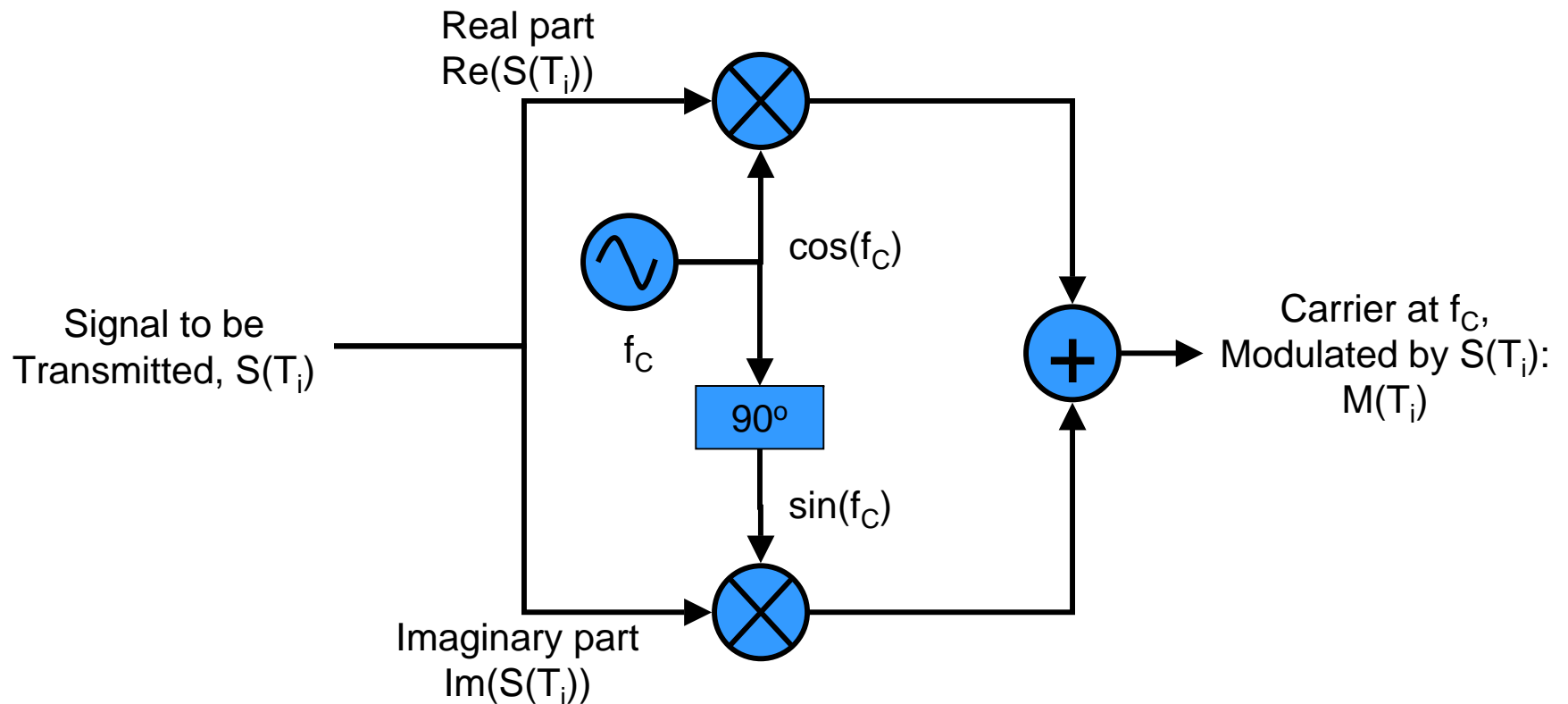


# Getting a Real-Time Embedded System Running



# 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;
    }
}
```

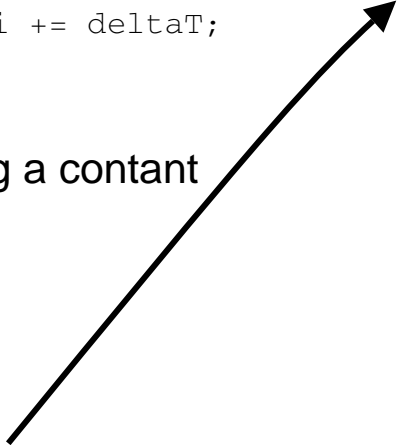
Real-time performance issues:

# 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

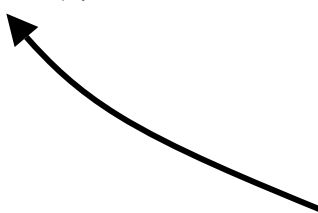
- Example code for a real-time embedded system:

```
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  $2\pi$  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);
```

$twopi*Ti*fc$  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;
    }
}
```

$twopi \cdot T_i \cdot fc$  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;
    }
}
```

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)
```

What is wrong with this code?

```
{
    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;
    }
}
```

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)
```

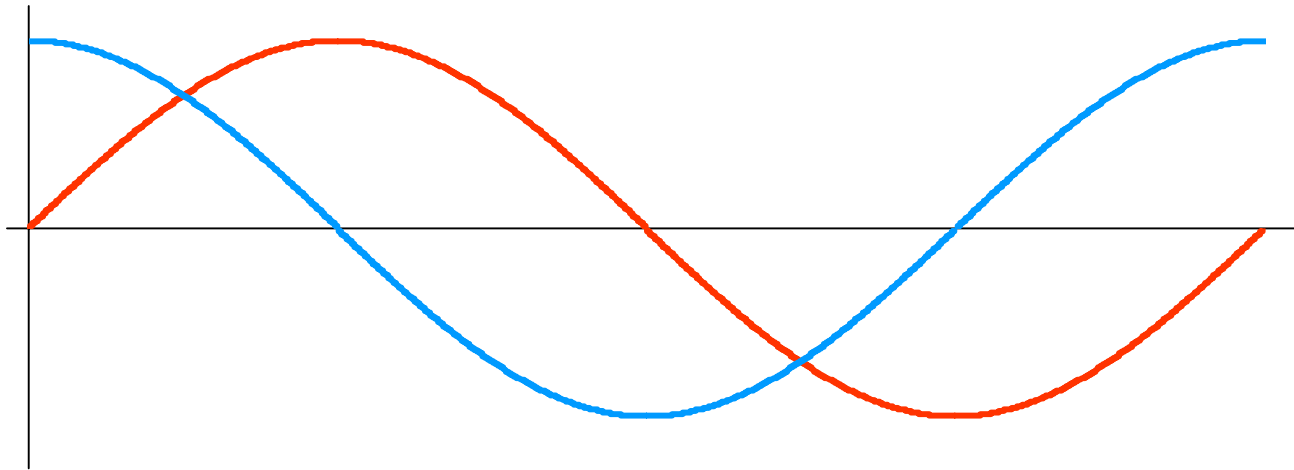
```
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase] +
              S[i].imag*sine_table[phase];
        phi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

What is wrong with this code?

How can memory and speed be improved?

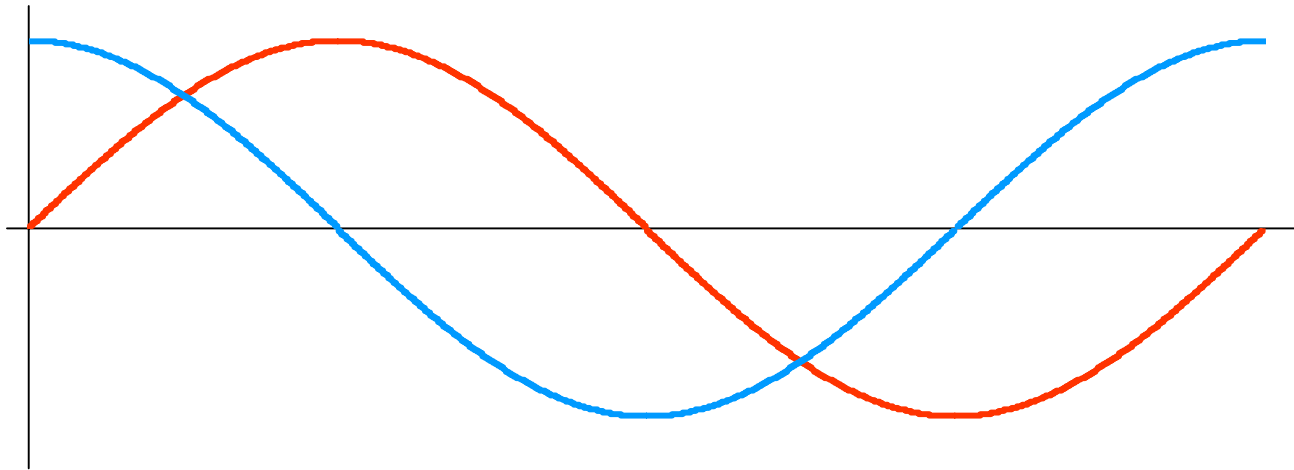
# Minimizing Sine Table Memory

- Consider the characteristics of sine and cosine:

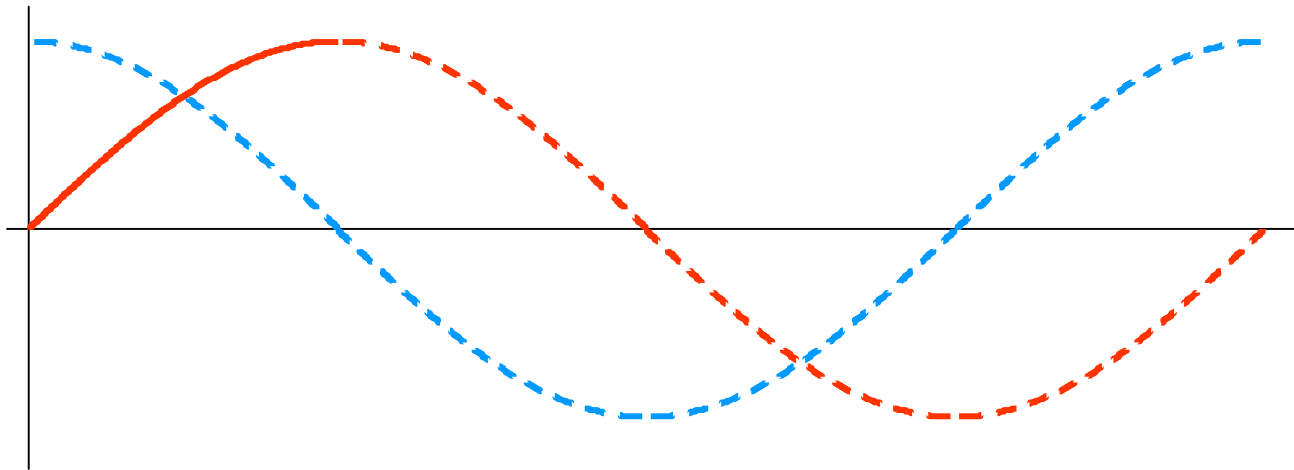


# Minimizing Sine Table Memory

- Consider the characteristics of sine and cosine:



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



# Assignment 7B

- 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.