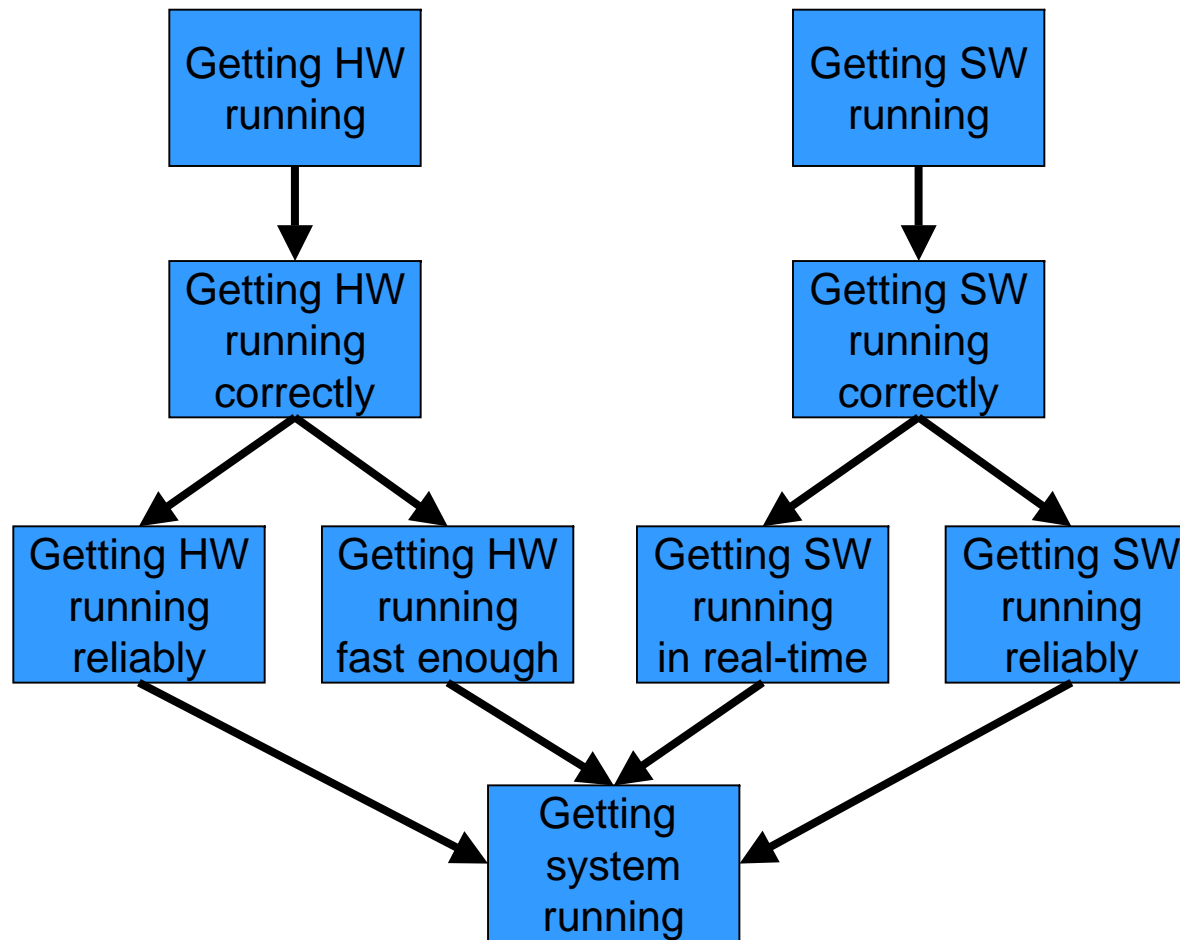


# Architecture, Design and Implementation of Embedded Systems for Real-Time Applications

CpE-450 - Spring 05  
Class 19

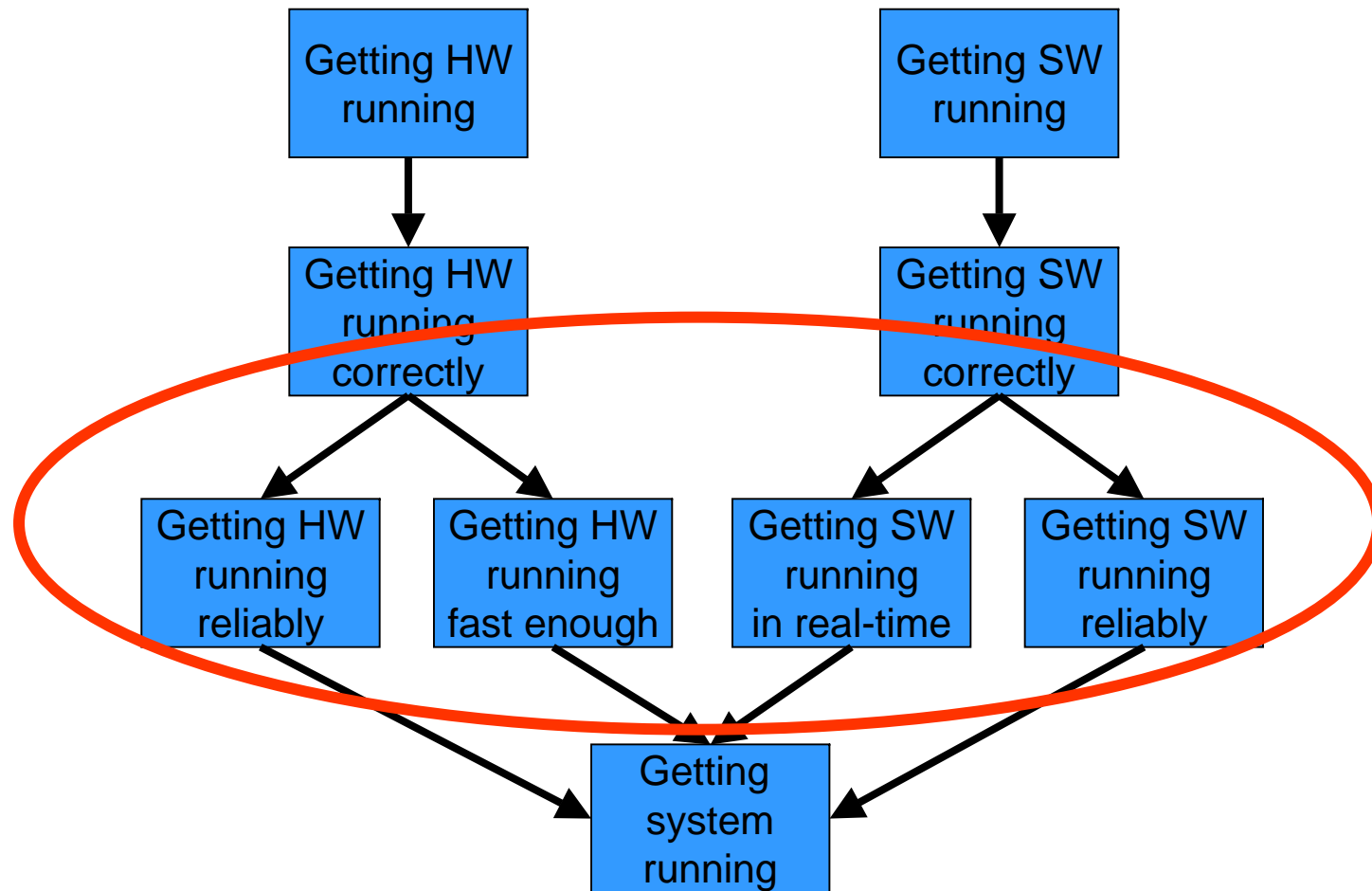
Bruce McNair  
bmcnair@stevens.edu

# Getting a Real-Time Embedded System Running



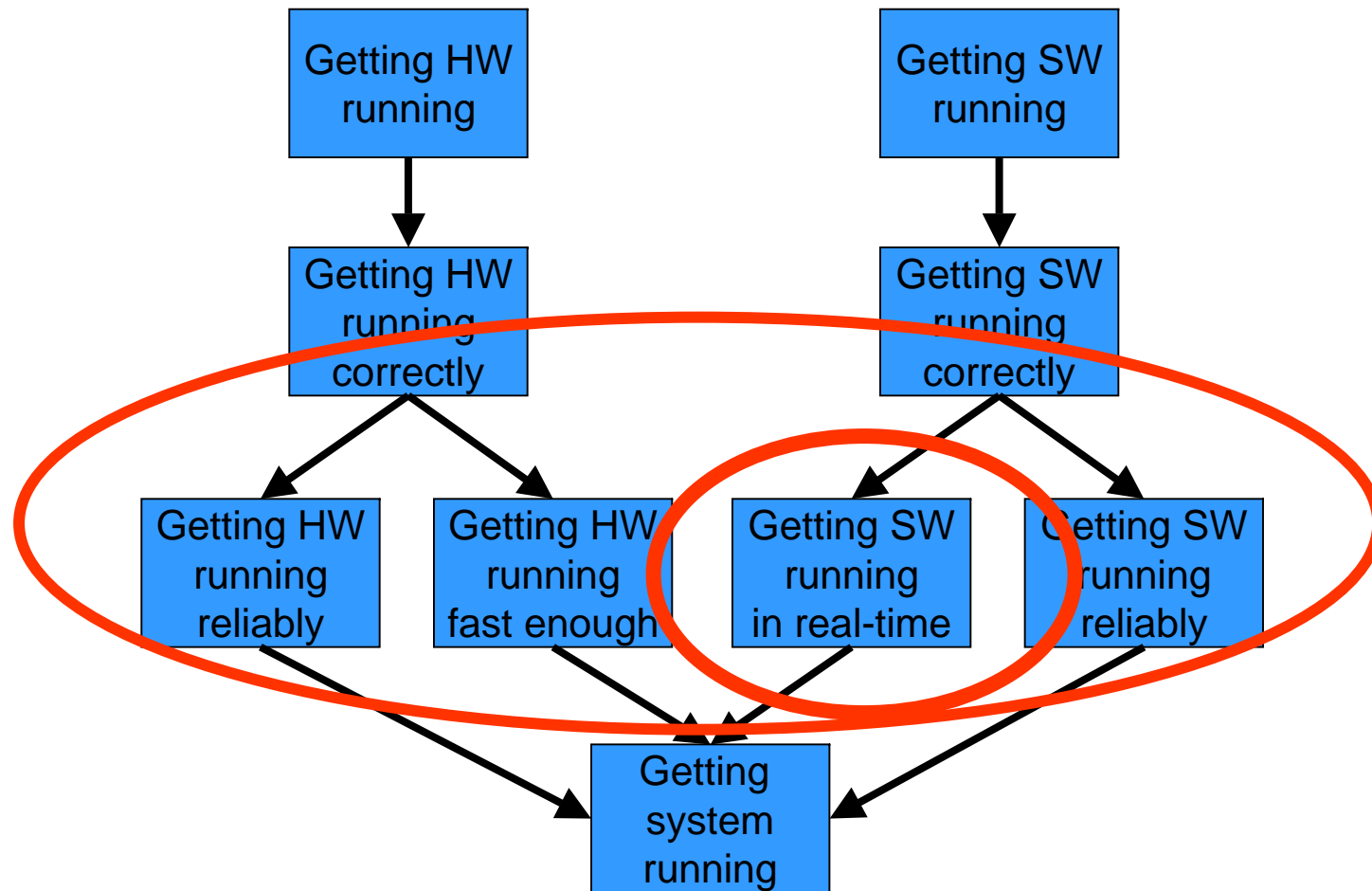
# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



# Getting a Real-Time Embedded System Running

- Evaluating performance and correctness



# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```

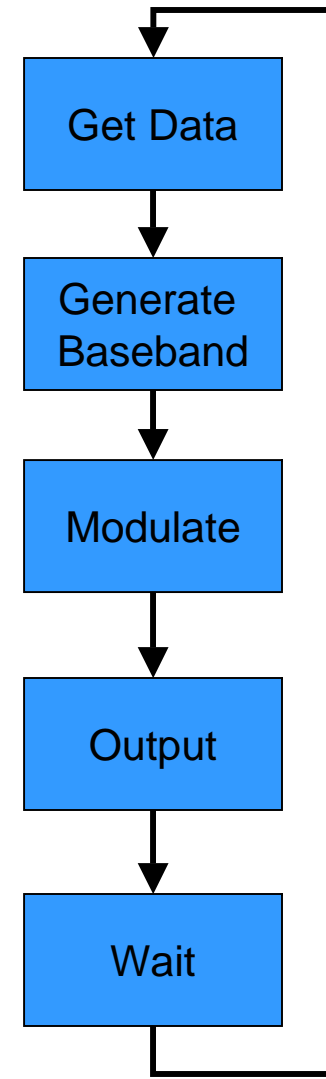
```
void modulate (signal *S, float *M, int N,
              float fc, deltaT)
{
    int i;
    static float phi=0;
    #define twopi=6.28318531
    float deltaPHI;
    deltaPHI=twopi*deltaT*fc;
    int phase;
    for(i=0; i<N, i++)
    {
        phase = (int) phi*SIZE/twopi;
        M[i] = S[i].real*cosine_table[phase]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

# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



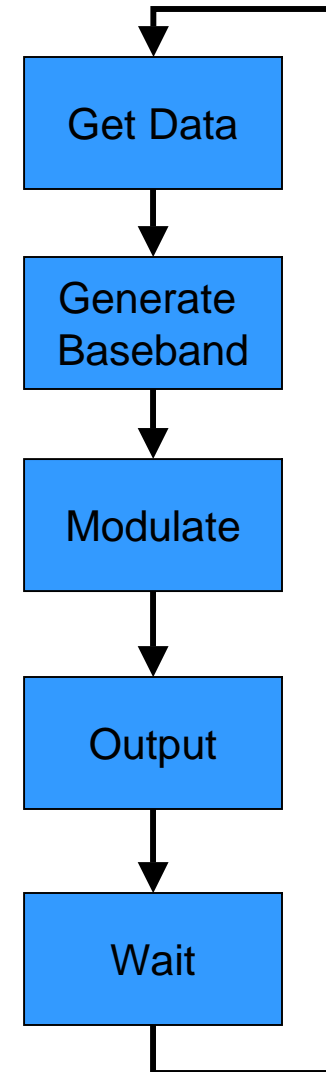
# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```

How can you tell if system is running in real-time?

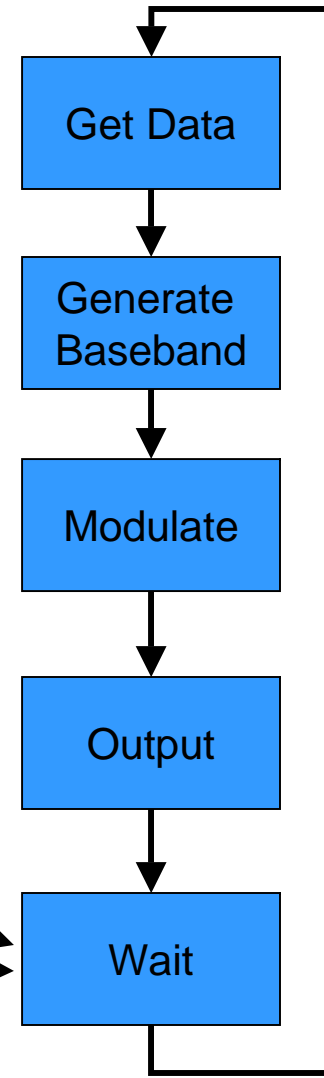


# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



Real-time:  $wait > 0$

Not Real-time: " $wait < 0$ "

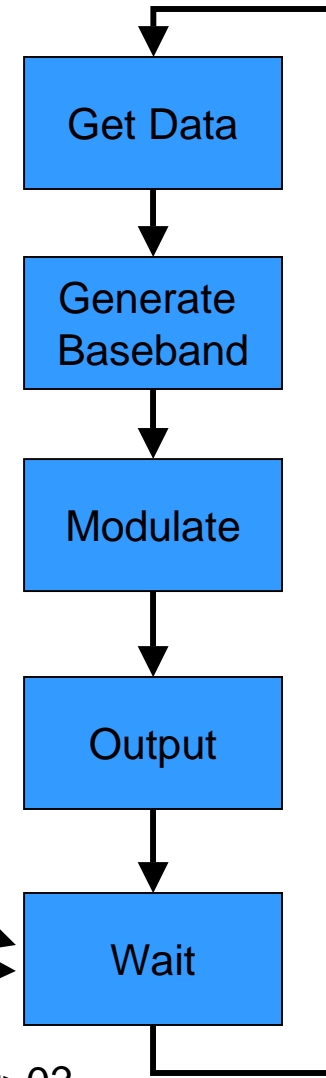
How can you tell if system is running in real-time?

# Assessing Real-Time Performance

- Polling

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        wait_for_next_frame( );
    }
}
```



Real-time:  $wait > 0$

Not Real-time: " $wait < 0$ "

How can you tell if system is running in real-time?

How do you know if  $wait > 0$ ?

# Assessing Real-Time Performance

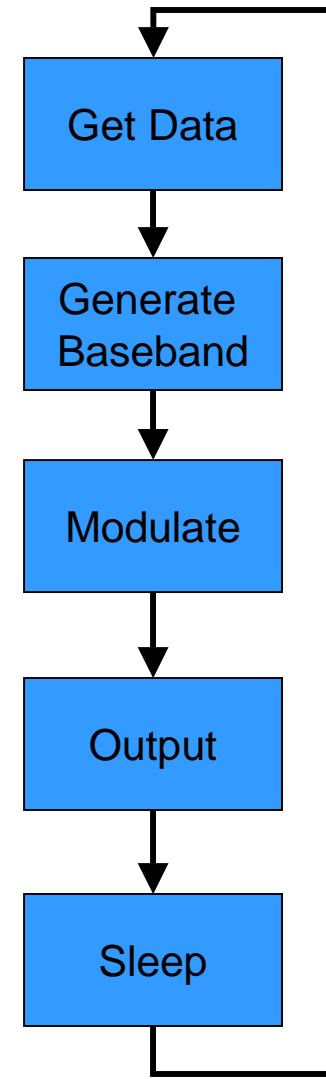
- ISR driven

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts (INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    /* wake up main( ) when data arrives */

```



# Assessing Real-Time Performance

- ISR driven

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};
```

How can you tell if system is running in real-time?

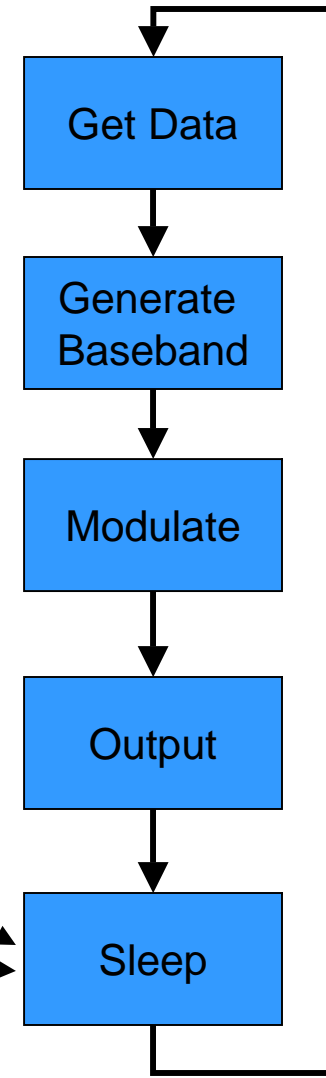
How do you know if sleeptime > 0?

```
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts (INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        sleep( );
    }
}

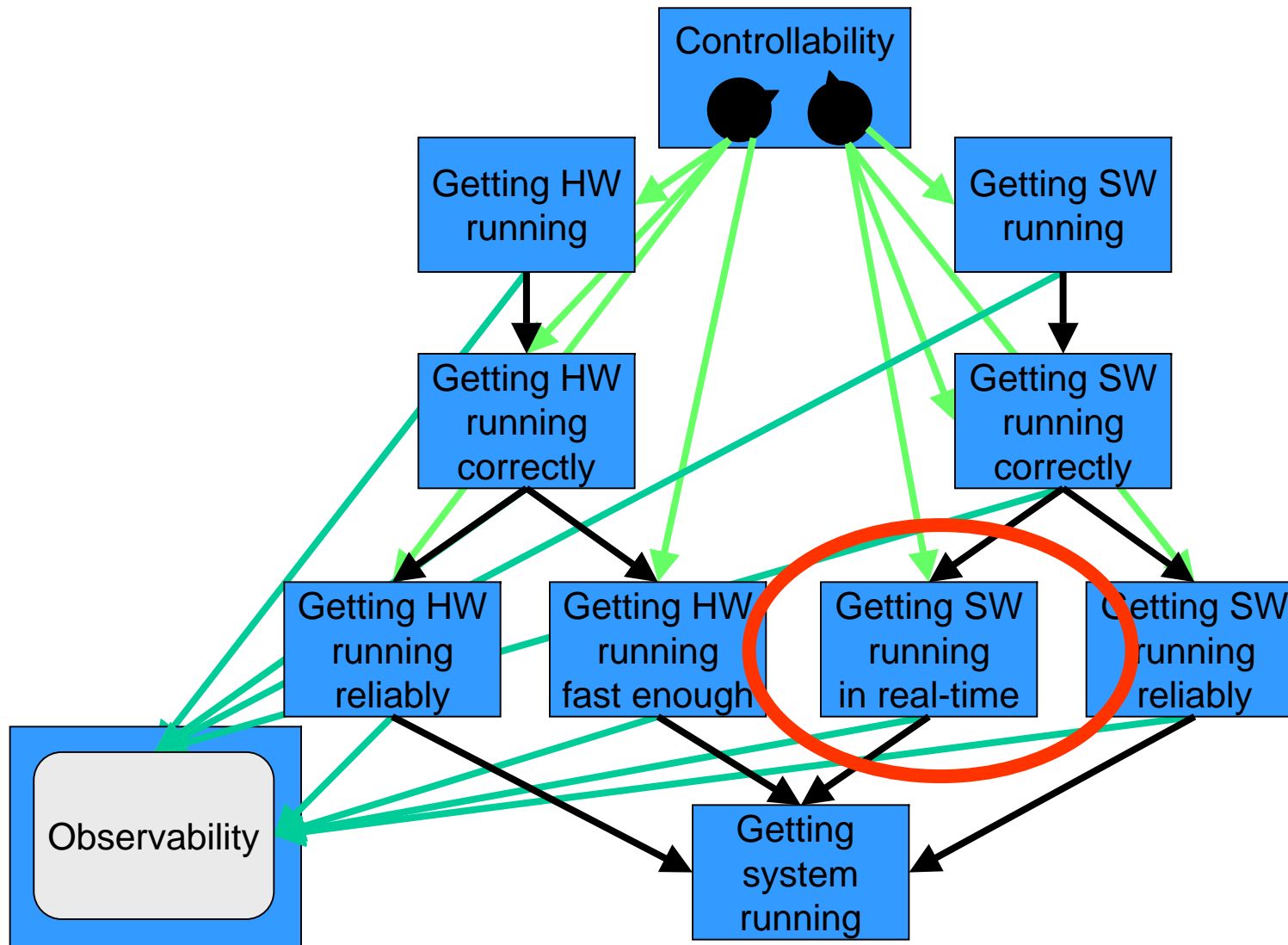
void interrupt_handler_DATA_PRESENT(void)
{
    /* wake up main( ) when data arrives */
```

Real-time: sleeptime > 0

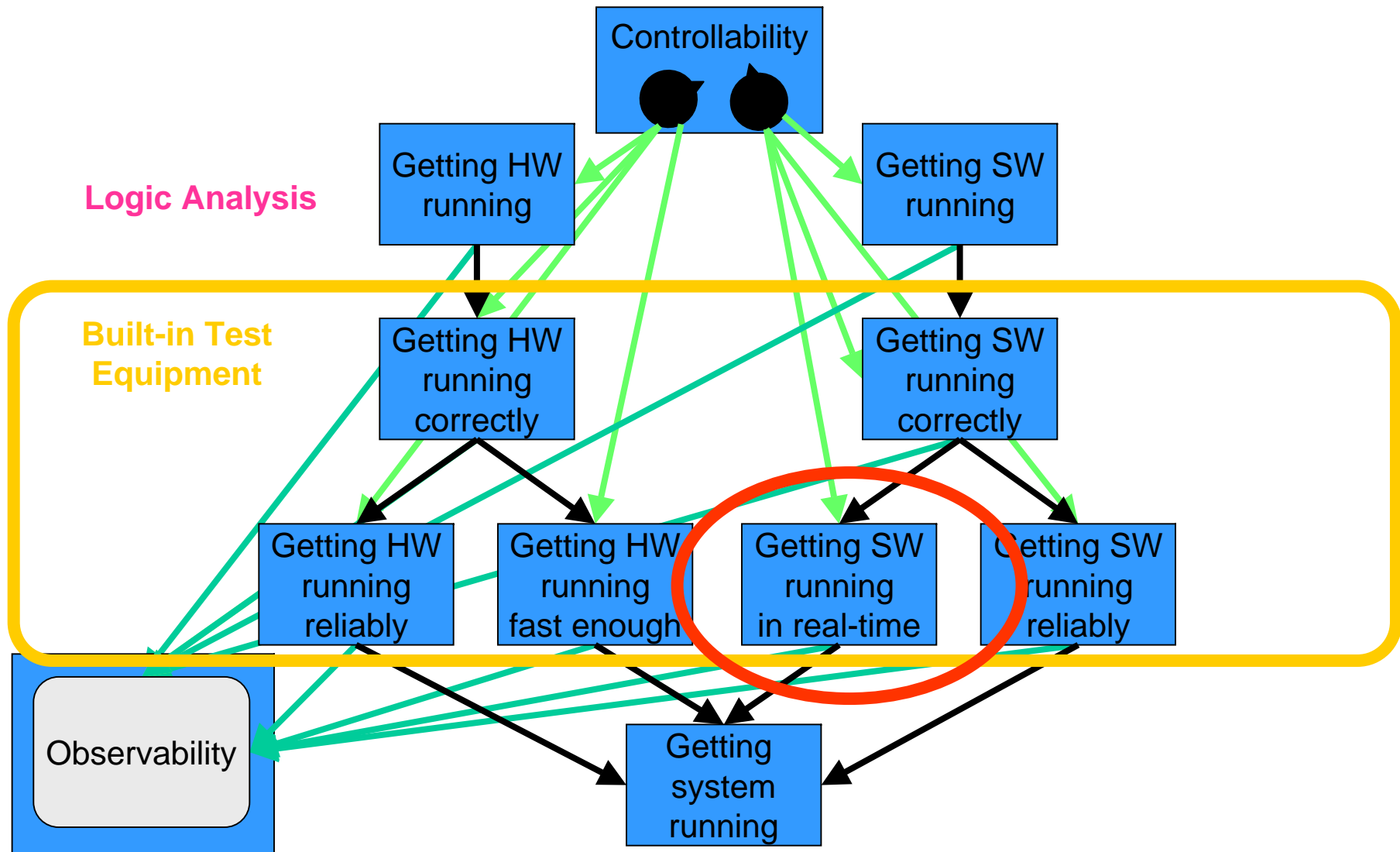
Not Real-time: "sleeptime < 0"



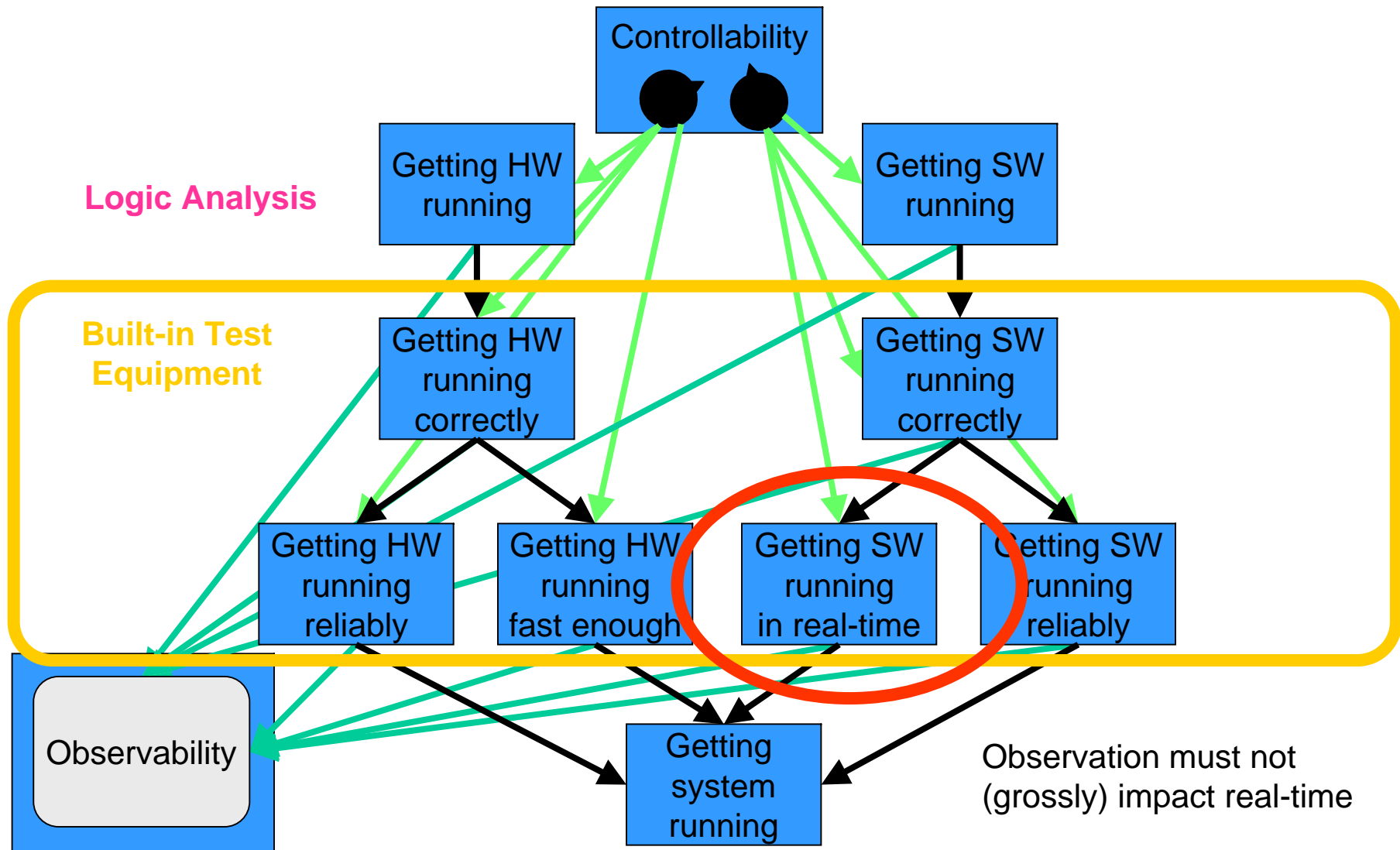
# Getting a Real-Time Embedded System Running



# Getting a Real-Time Embedded System Running



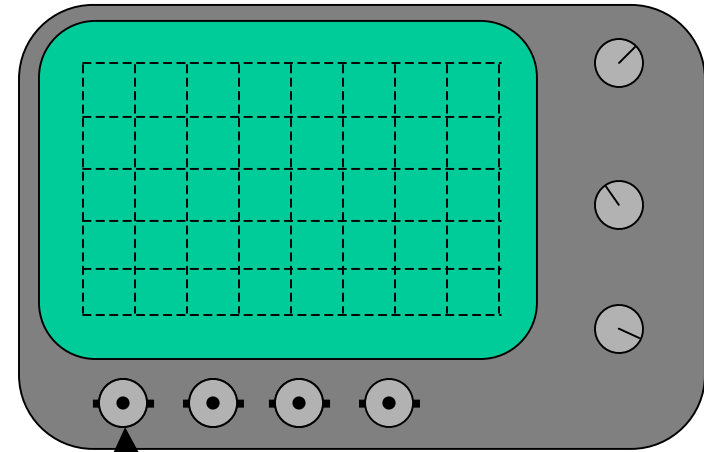
# Getting a Real-Time Embedded System Running



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

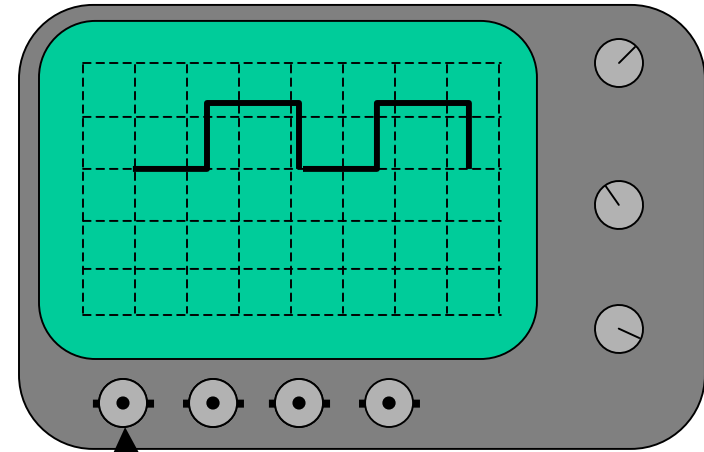
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

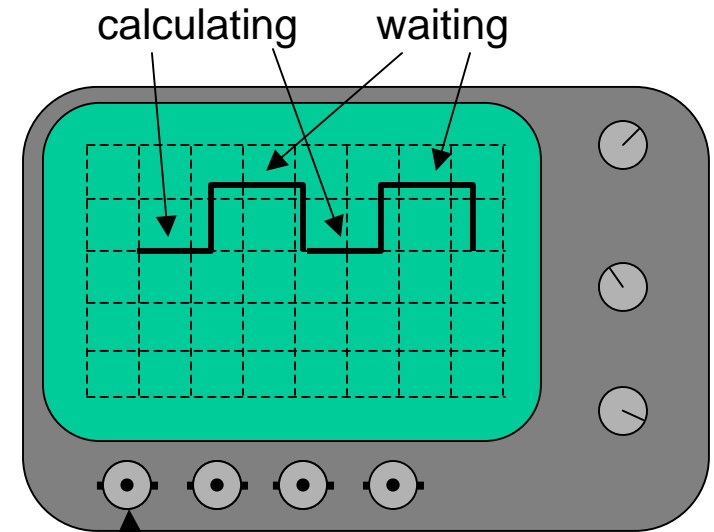
void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```

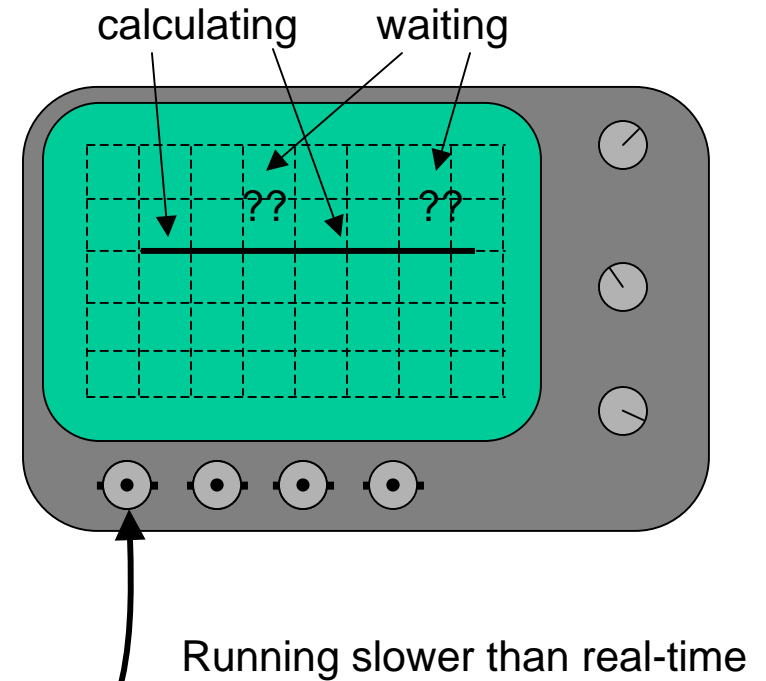


Running faster than real-time

# Real-time Observability

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        output_mod(mod);
        set_output_flag(1);
        wait_for_next_frame( );
        set_output_flag(0);
    }
}
```



# Real-time Observability

- Observing time spent in ISR, other functions

```
float sine_table[SIZE];
float cosine_table[SIZE];
struct signal
{
    float real;
    float imag;
};

void main( )
{
    boolean bits[BLOCK_LENGTH];
    signal sig[BLOCK_LENGTH];
    float mod[BLOCK_LENGTH];
    initialize_sine_table(*sine_table, *cosine_table);
    set_up_interrupts (INTERRUPT_ON_DATA_PRESENT);
    while(1)
    {
        get_data(bits);
        generate_baseband(bits, sig);
        set_output_flag(1);
        modulate(sig, mod, BLOCK_LENGTH, Fc, delta_t);
        set_output_flag(0);
        output_mod(mod);
        sleep( );
    }
}

void interrupt_handler_DATA_PRESENT(void)
{
    set_output_flag(1);
    /* wake up main( ) when data arrives */
    set_output_flag(0);
}
```

