Yamacraw Research Program in Embedded Software

# A Debugger RTOS for Embedded Systems

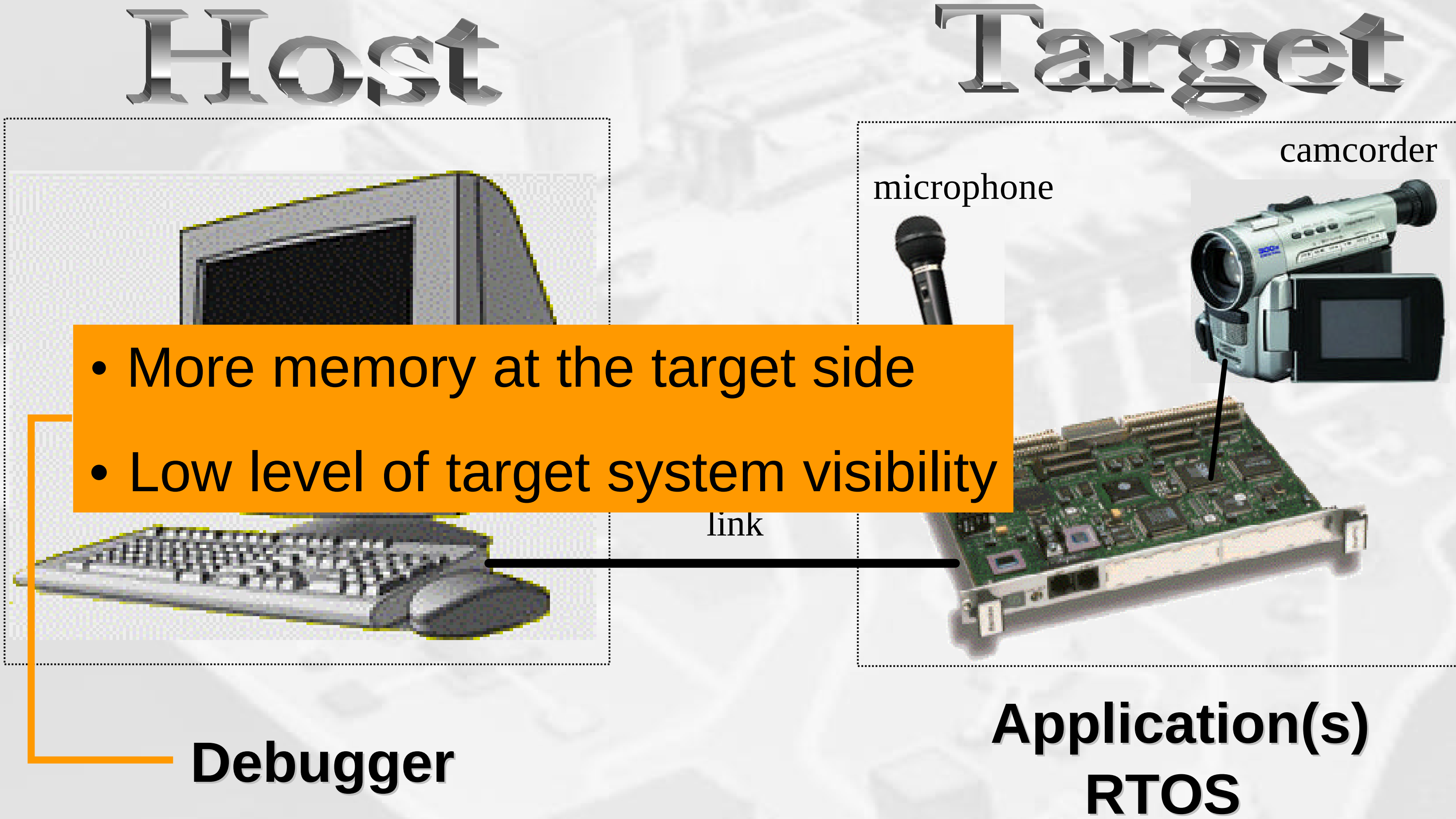
**Faculty : Vincent Mooney, Vijay Madisetti**
**Students : Tankut Akgul, Pramote Kuacharoen**

yamacraw futureforward
Advancing Electronic Design in Georgia

# Outline

- Background and Motivation
- Approach
- Architecture
- Example
- Conclusion

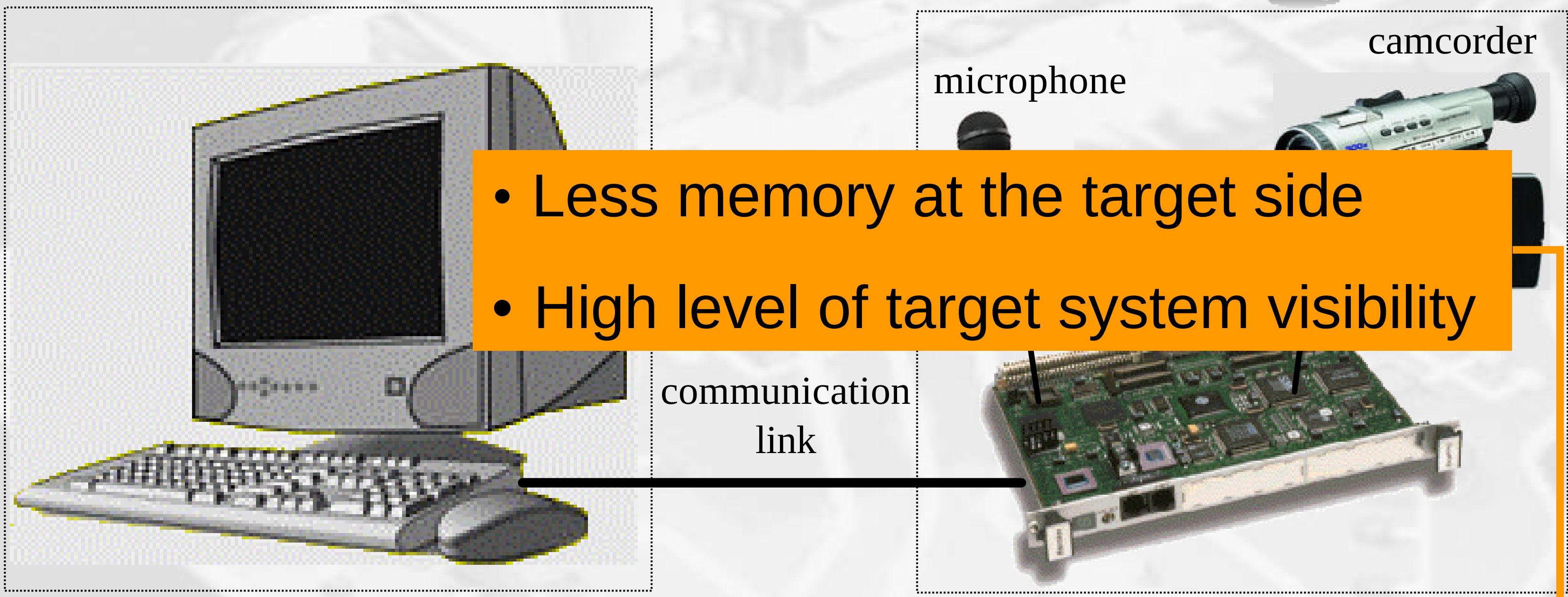
# Background and Motivation




- More memory at the target side
- Low level of target system visibility

# Background and Motivation

**Host** **Target**



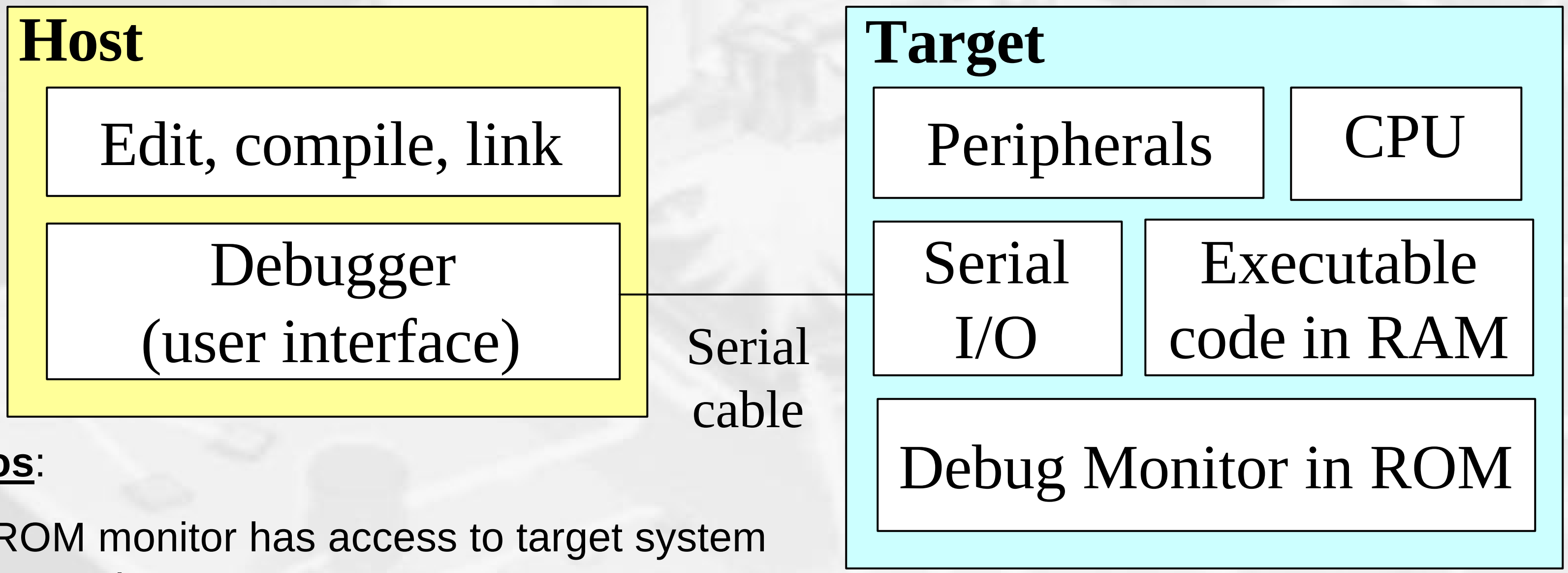

- Less memory at the target side
- High level of target system visibility

**Application(s)**
**RTOS**
**Debugger**

yamacraw
futureforward

# Background and Motivation

## ROM (Debug) Monitor




**Pros**:

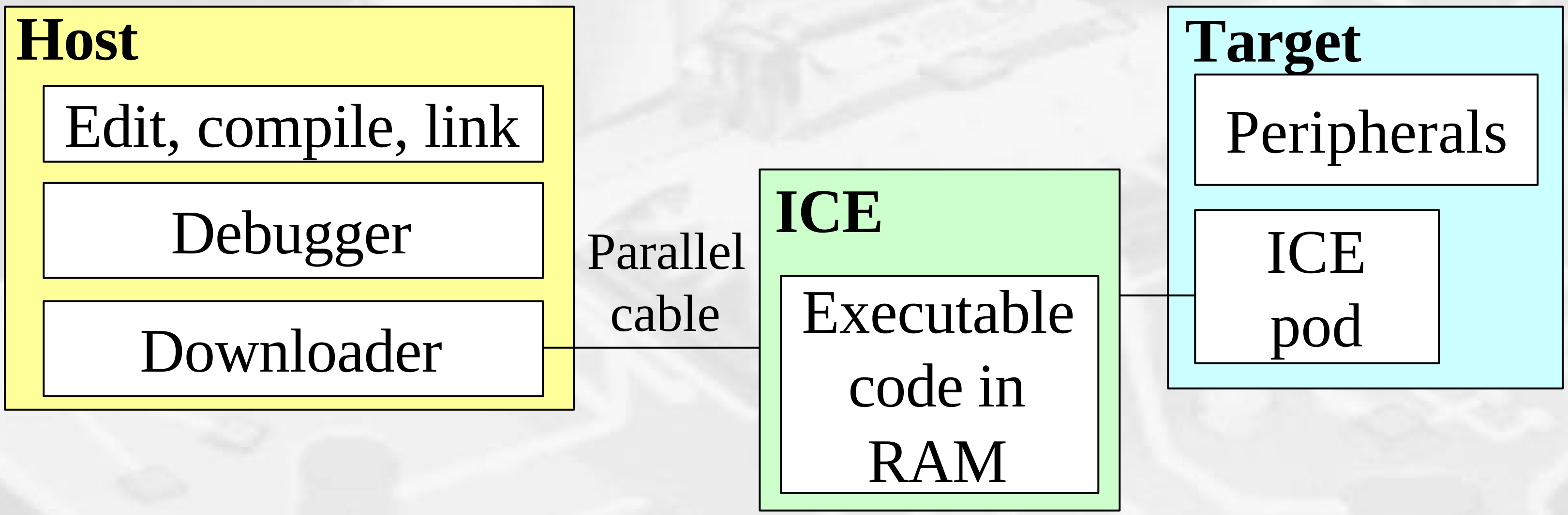
- ROM monitor has access to target system internals

**Cons**:

- Hard to remove the ROM monitor from the final product
- Fixed monitor code once burned into ROM

# Background and Motivation

## In-Circuit-Emulator (ICE)




**Pros**:

- Real-time event detection, real-time tracing

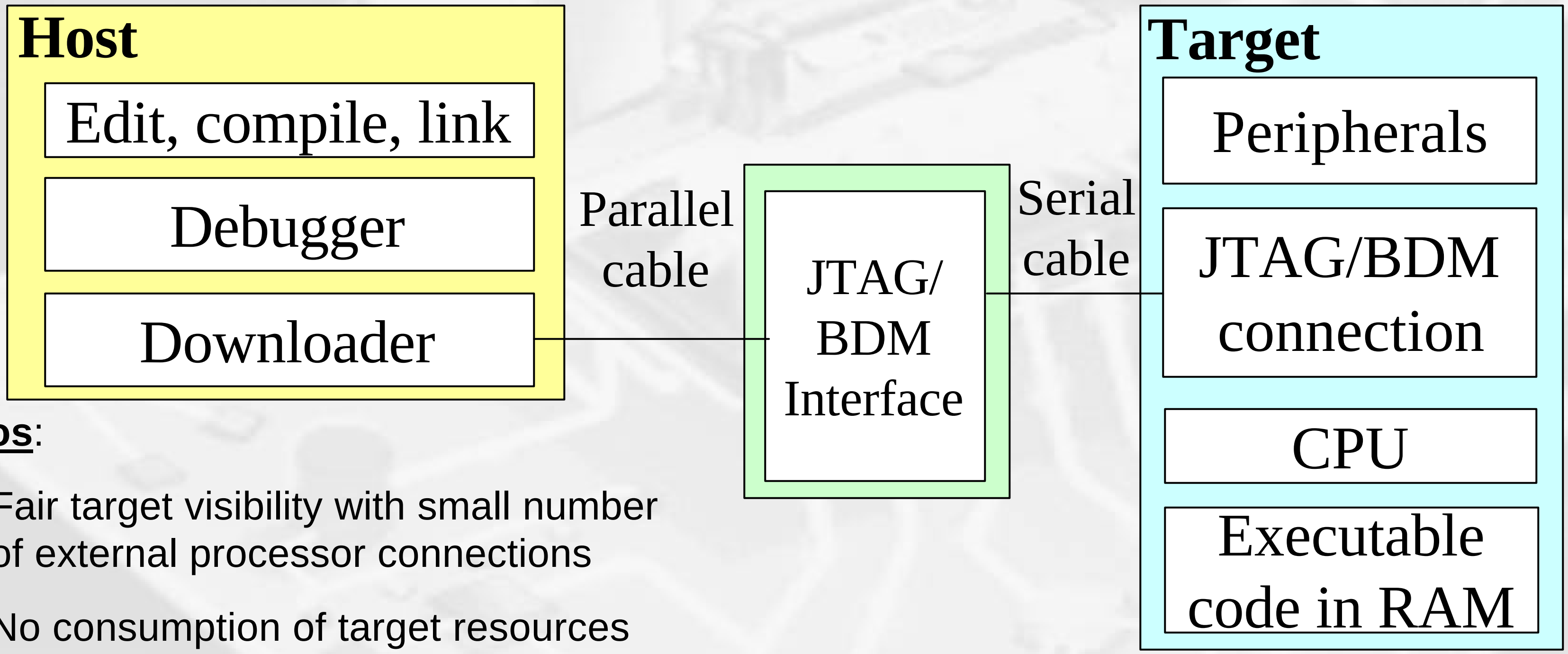
**Cons**:

- Expensive
- Lags processor production time
- Does not work well with high speed signals
- Impractical to use with highly embedded cores

yamacraw futureforward

# Background and Motivation

## JTAG/Background Debug Mode (BDM)



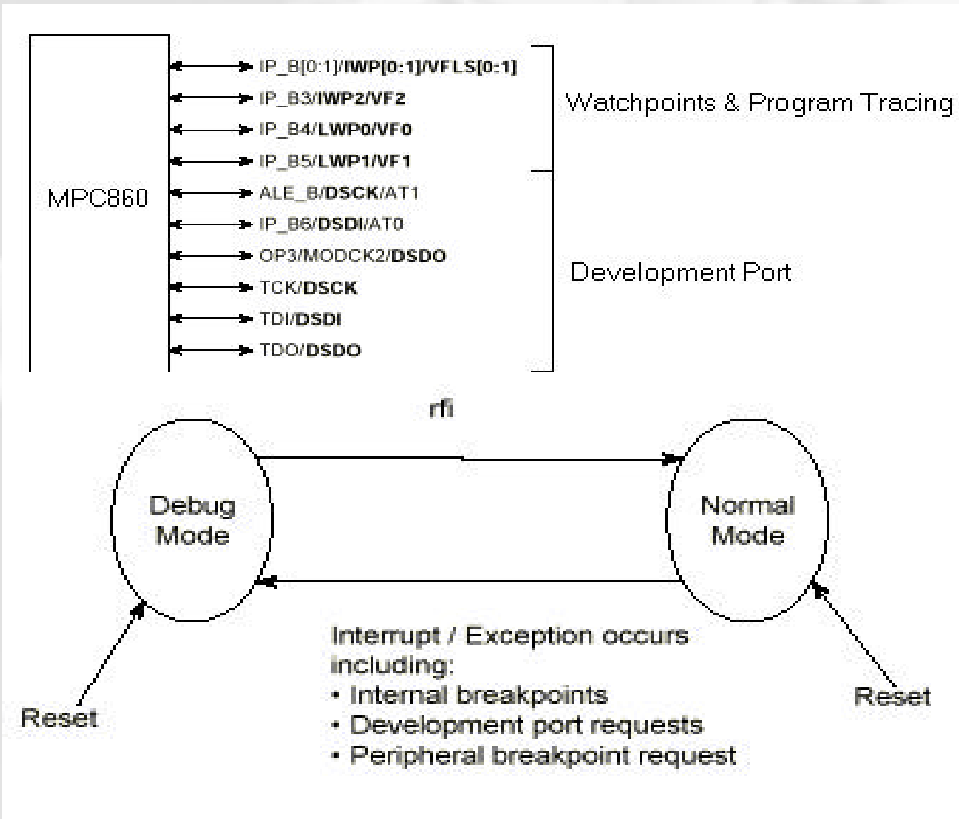

<u>**Pros**</u>:

- Fair target visibility with small number of external processor connections
- No consumption of target resources

<u>**Cons**</u>:

- Requires on-chip hardware support
- Not suitable for real-time debugging

# Example 1: BDM with MPC860




- User can set at most 5 watchpoints in software. Whenever these watchpoints are reached, external pins are asserted.
- The processor, if programmed, goes into the debug mode on exceptions, and stops execution. Then, a remote debugger can send commands to the processor through the processor's external pins.

# Example 2: Incorrect Program Trace

```
…
If (x>0) {
  ! sum = sum + x;
printf("x is positive");
….
}
else {
    sum = sum – x;
printf("x is negative");
….
}
```

Breakpoint inserted (at `! sum = sum + x;`)

- Branch prediction hardware predicts "x>0",
- Pre-fetcher fetches instruction here,
- Traditional host debugger monitors address of this instruction on external signals, breaks there,
- Prediction is wrong, branch not taken $\Rightarrow$ trace is incorrect !
- On-chip debugger has full knowledge of internal signals, this misbehavior never happens!

# Approach

## Just-in-time Debugging

- Debugger software is a loadable module of the RTOS
- Buggy code runs on the processor until an exception condition or an algorithmic error occurs.
- Debugger is dynamically loaded by the exception handler.

## Automatic Error Detection

- **Algorithmic errors**
  - E.g., functional errors, timing errors, synchronization errors.
- **Hardware Exceptions**
  - E.g., alignment errors, software emulation error.

# Approach

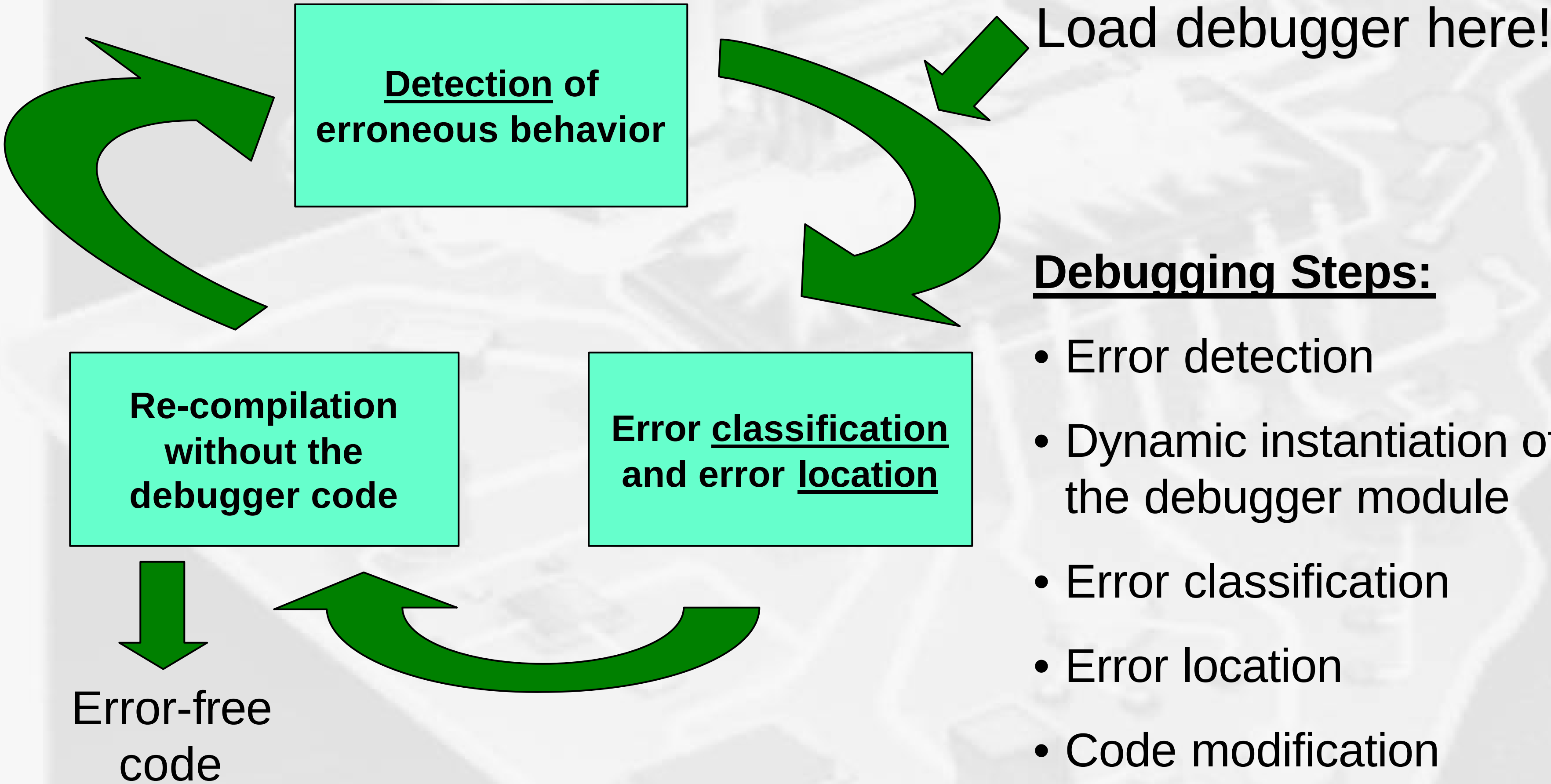
## Error Monitor

- Always resident inside RTOS
- Responsible for the detection of errors and the instantiation of the debugger part

## Debugger Module

- Responsible for classifying, and locating errors in the program
- Provides debugging features such as memory dump, and register display

# Approach




**Debugging Steps:**

- Error detection
- Dynamic instantiation of the debugger module
- Error classification
- Error location
- Code modification
- Code re-compilation

yamacraw futureforward

# Future Improvements

- Current features of the debugger module:
  - Error classification
  - Error location for immediately locatable errors
  - Display of register values
- Future improvements:
  - Incremental instantiation of additional features
  - Breakpointing
  - Source line stepping
  - On the fly variable query and modification
  - Profiling

# Architecture

## The Georgia Tech Dynamic RTOS

Modules

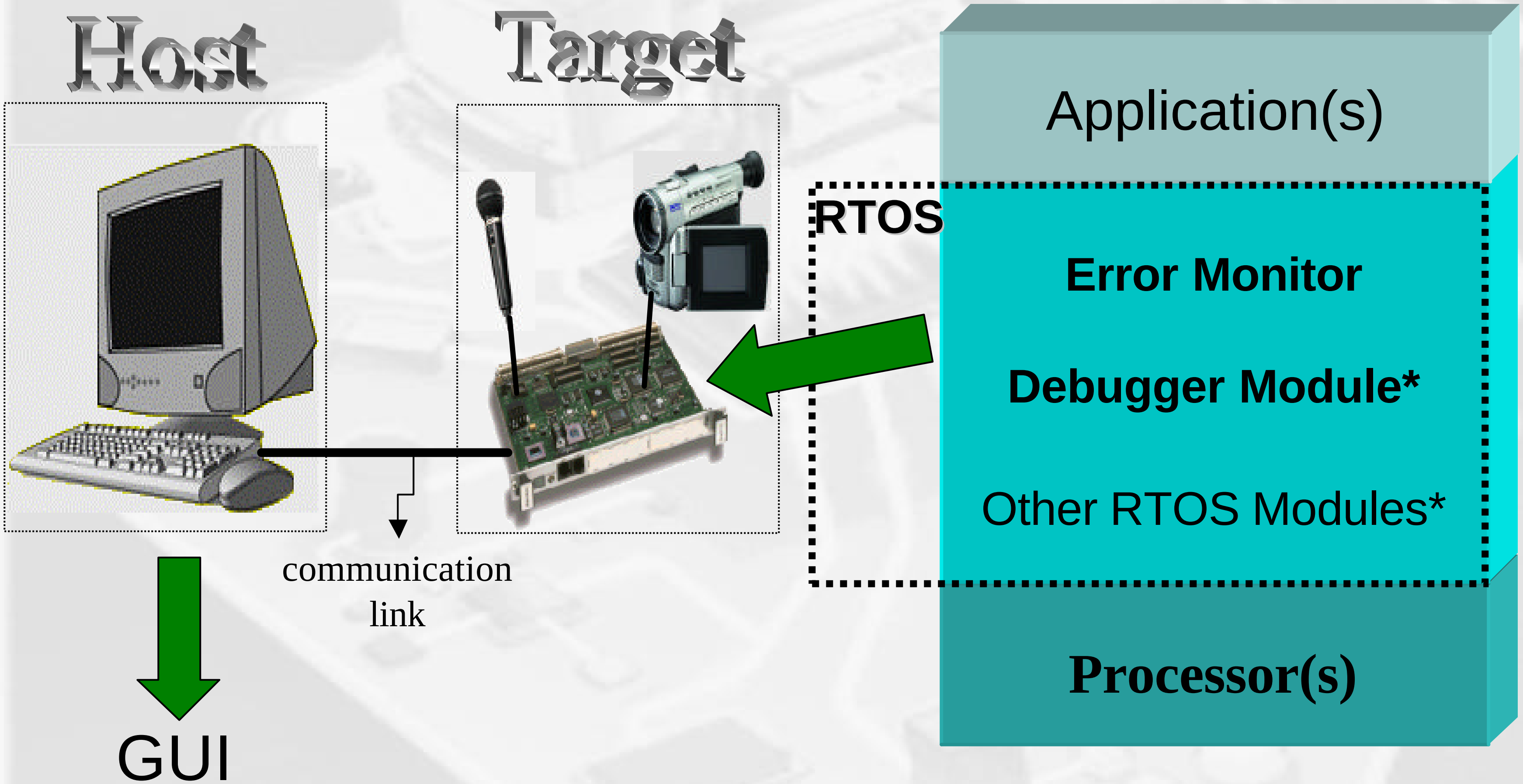
| | | | |
|---|---|---|---|
| Core | Time | Task | Scheduler |
| Memory | Comm | Debugger | Loader |

| Core Module | |
|---|---|
| Text | Executable code |
| Data | Module Variables<br>System APIs<br>Core<br>Time<br>Task<br>Scheduler<br>Memory<br>Communication<br>Debugger<br>Loader |

Reference:

P. Kuacharoen, T. Akgul, V. J. Mooney, V. K. Madisetti, "Adaptability, Extensibility, and Flexibility in Real-Time Operating Systems", *Euromicro Symposium on Digital System Design*, September 2001.

# Architecture




* Dynamically loadable

yamacraw
futureforward

# Architecture

Module Table

| Function name | Address |
| --- | --- |
| Debugger | 0xFF000000 |
| Scheduler | 0xFA000000 |
| … | … |

Global Variable Table

| Global var. | Address |
| --- | --- |
| cur_task_ID | 0xFF00A000 |
| TCB_list | 0xFFF00B00 |
| … | … |

***load_module_debugger() {***

***1***

***}***

***init_module_debugger() {***

***2 , 3***

***}***

1 **Load debugger module into memory**

2

0xFF000000

3

Debugger Module

***Text Section***

***Data Section***

Local Variables

Global Variable pointers

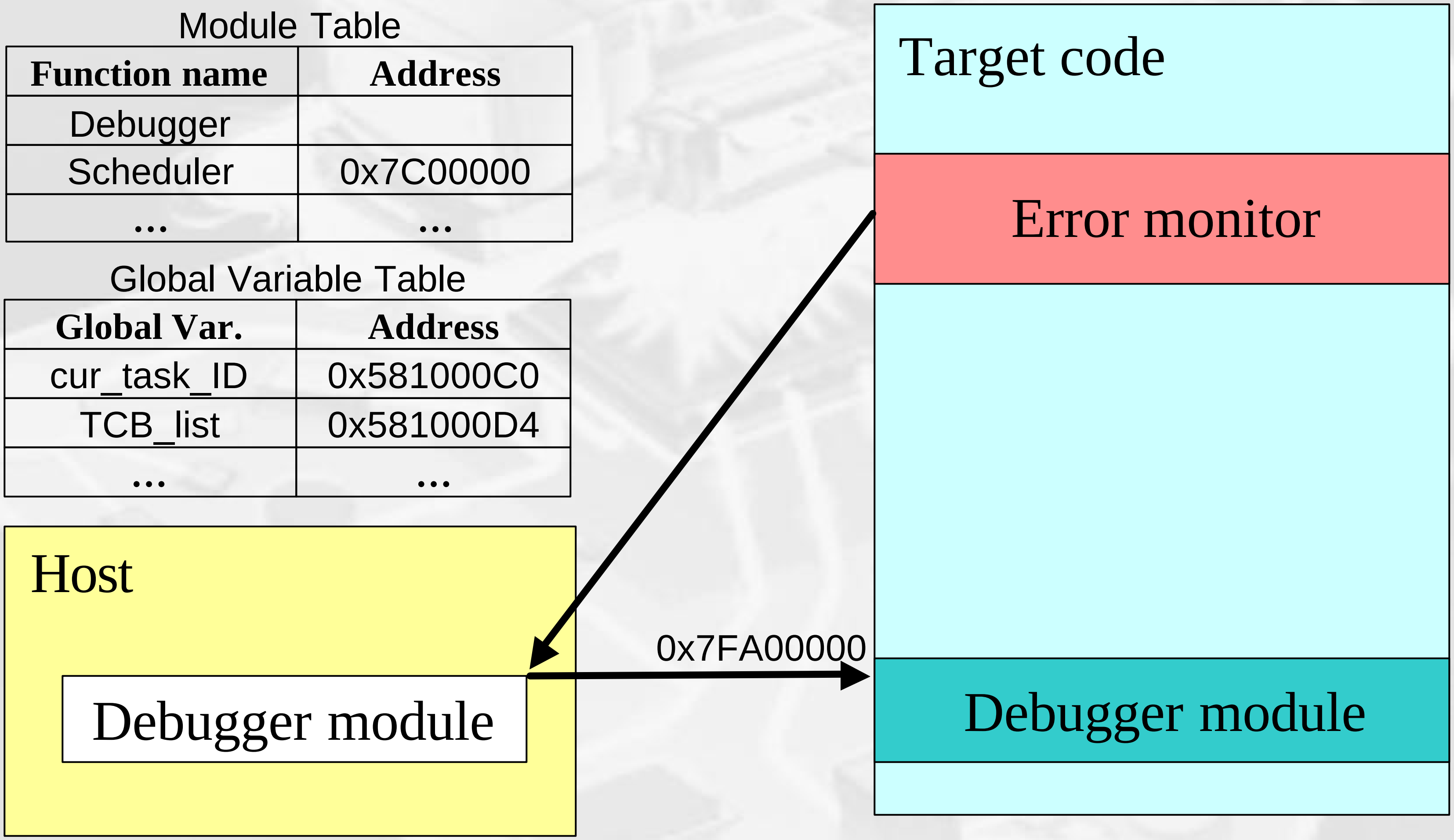
# Example

Module Table

| Function name | Address |
|---|---|
| Debugger | |
| Scheduler | 0x7C00000 |
| … | … |

Global Variable Table

| Global Var. | Address |
|---|---|
| cur_task_ID | 0x581000C0 |
| TCB_list | 0x581000D4 |
| … | … |



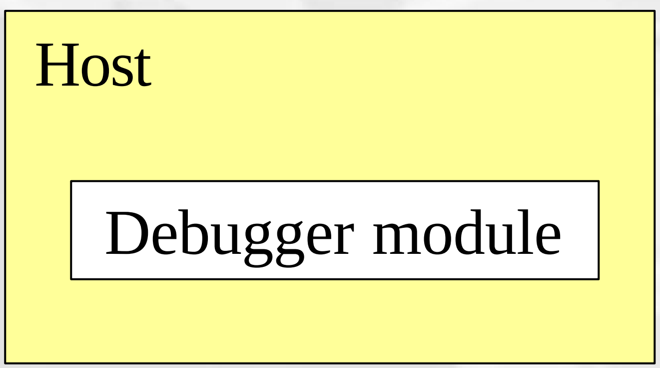
# Example

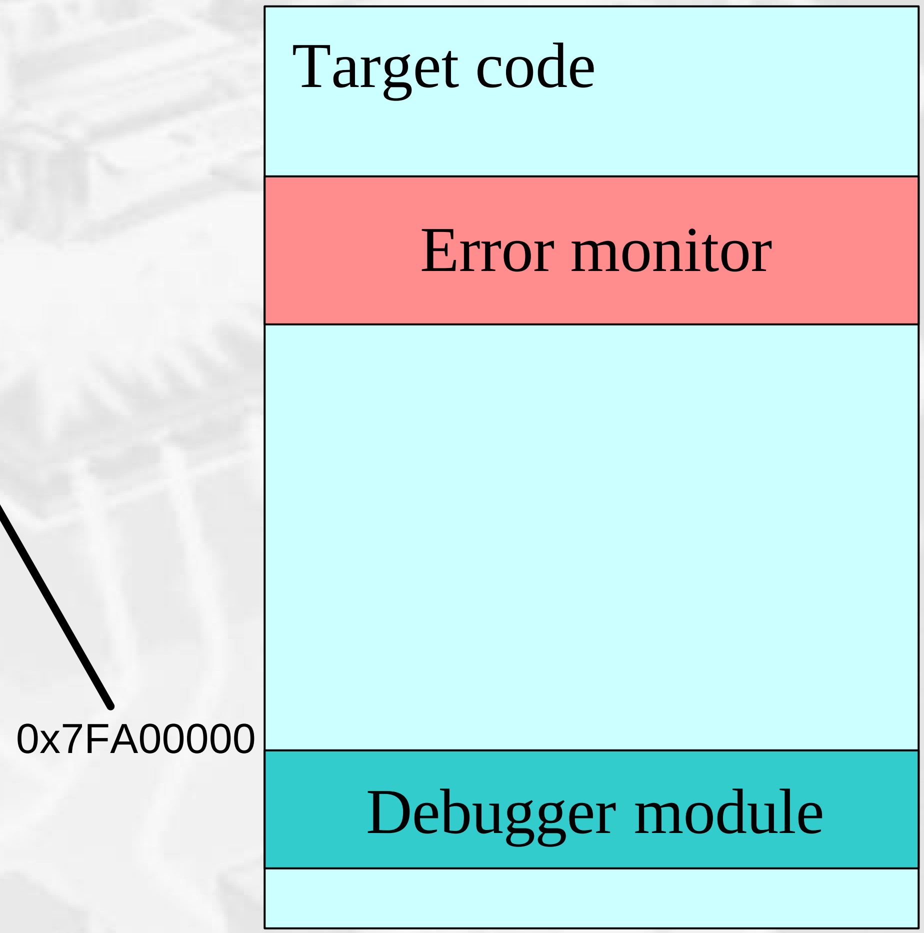
Module Table

| **Function name** | **Address** |
|---|---|
| Debugger | 0x7FA00000 |
| Scheduler | 0x7C00000 |
| ... | ... |

Global Variable Table

| **Global Var.** | **Address** |
|---|---|
| cur_task_ID | 0x581000C0 |
| TCB_list | 0x581000D4 |
| ... | ... |



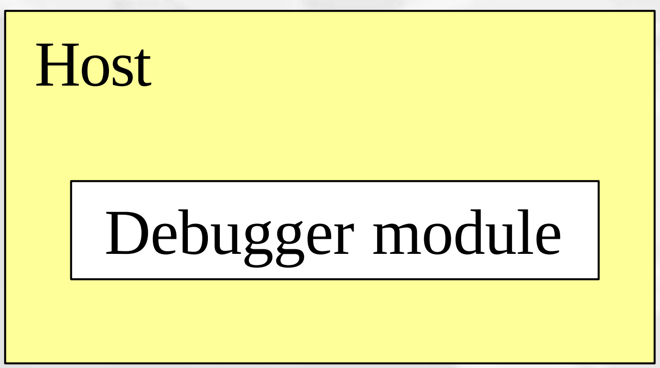
# Example

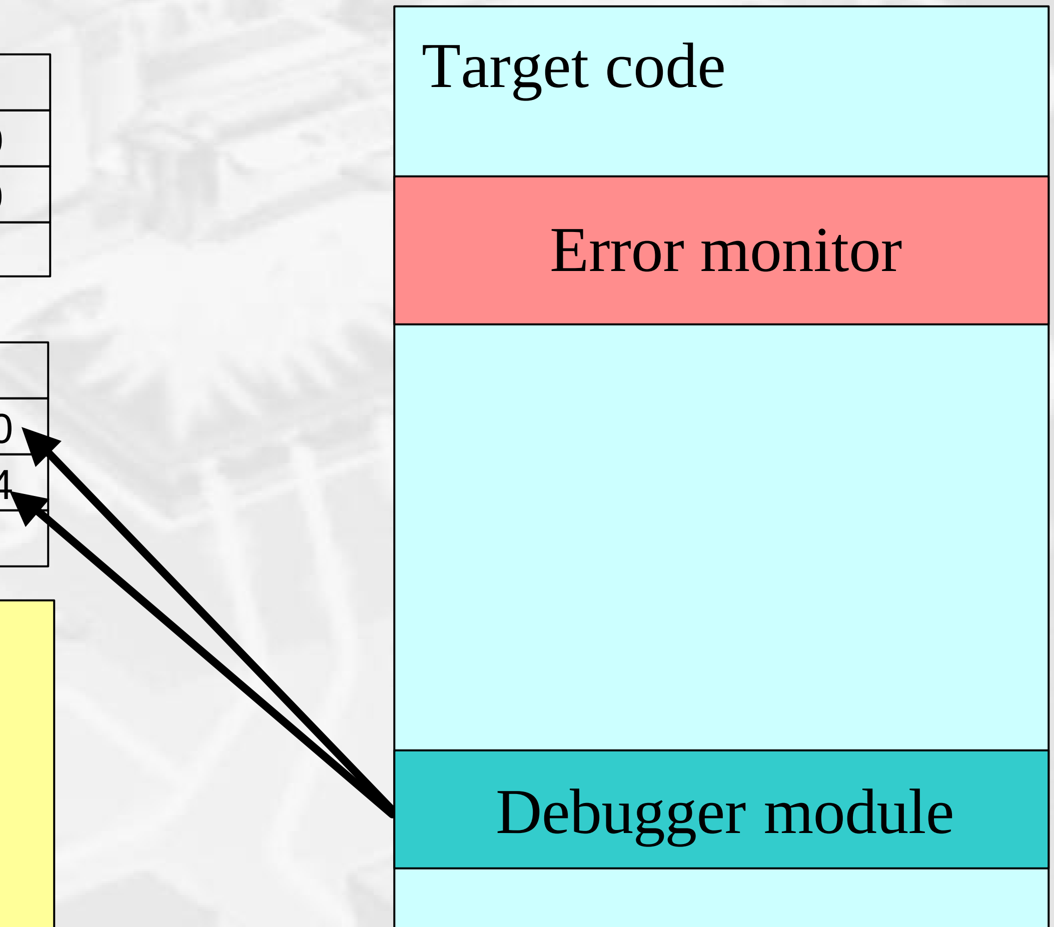
Module Table

| Function name | Address |
|---|---|
| Debugger | 0x7FA0000 |
| Scheduler | 0x7C00000 |
| ... | ... |

Global Variable Table

| Global Var. | Address |
|---|---|
| cur_task_ID | 0x581000C0 |
| TCB_list | 0x581000D4 |
| ... | ... |



# Example Application

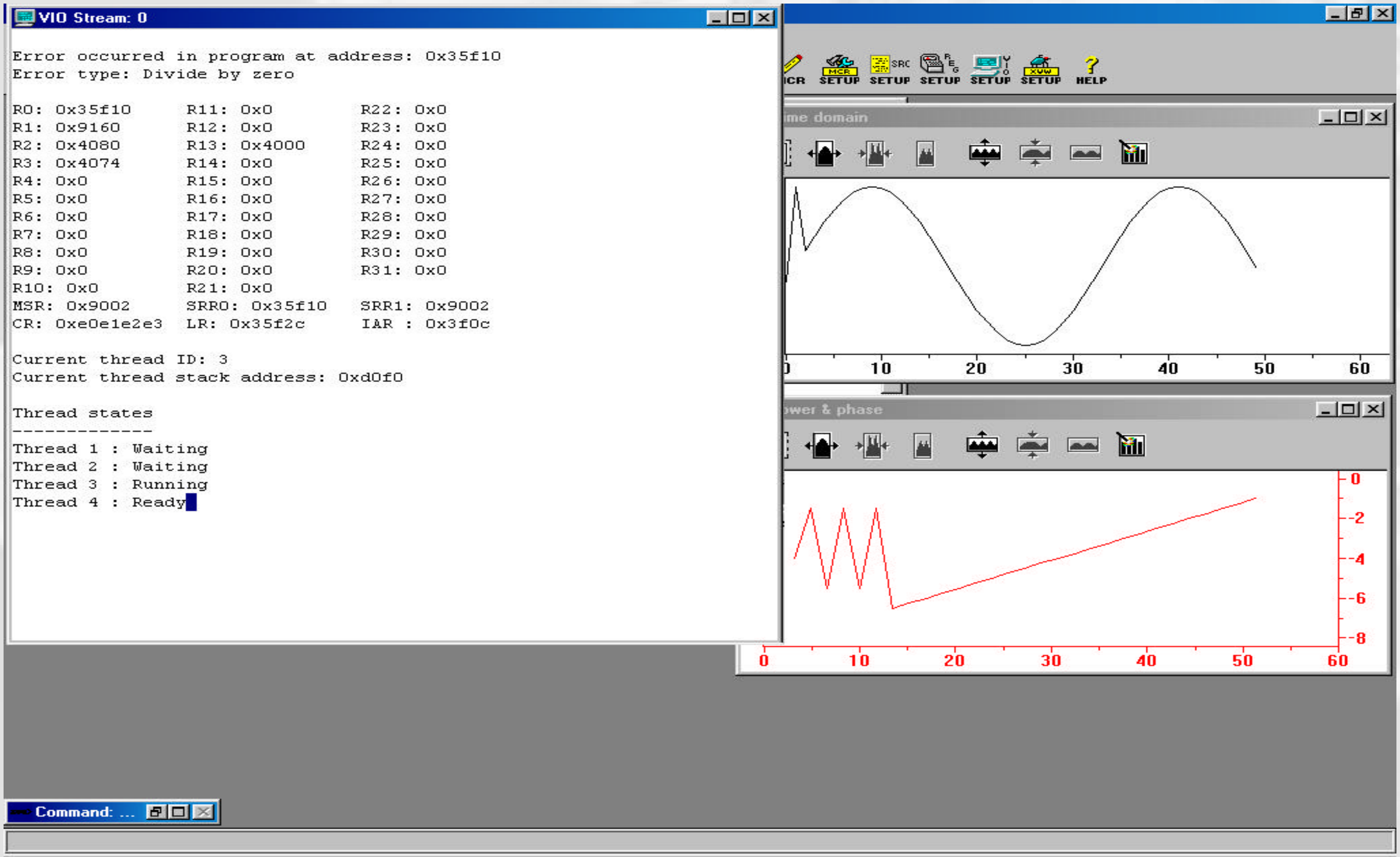
- Repeatedly gets two floating point voltage samples from two different locations in memory
- Converts collected voltage samples into integer values
- Calculates the ratio of the samples

```
int find_ratio(v1, v2)
{
  v1 = (int) get_voltage(loc1);
  v2 = (int) get_voltage(loc2);
  return (v1/v2);
}
```

→ **Creates divide-by-zero algorithmic error* when v2 = 0**

*** MPC860 does not have an exception for the integer divide-by-zero error.**

# A Snapshot of the GUI



# Conclusion

- ☑ Efficient Memory Usage
- ☑ Capability for detecting algorithmic errors
- ☑ No extra hardware for debugging
- ☑ Correct debugging in a highly coupled and parallel environment