

# Signal Processors

## Lecture Notes, Pt 3

Institute for Technical Informatics, DI Dr. Eugen Brenner Signal Processors, Pt 3

03.04.2018

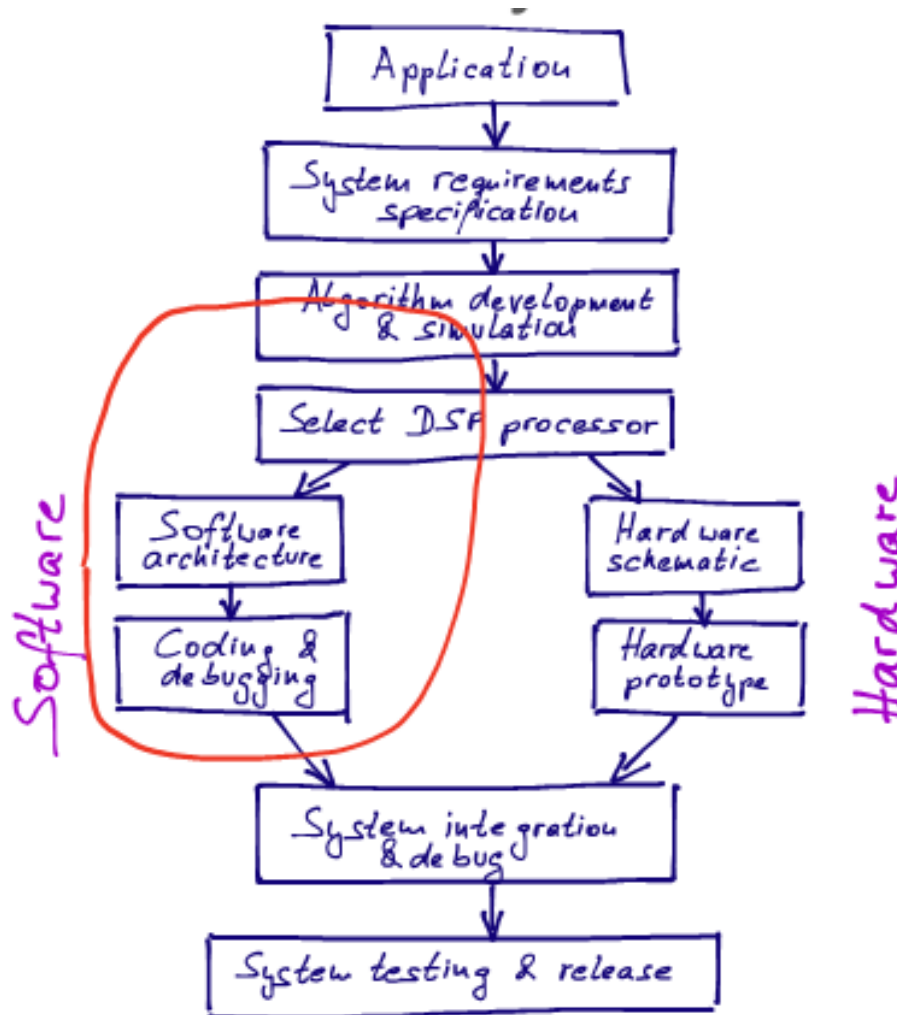
# Chapter Overview

## 3. Development of DSP Systems

- DSP Software Development Process
- Design Challenges
- Modelling of DSP Systems
- Algorithm and Architecture
- Software Architecture – DSP/BIOS
- Conclusion

# DSP Software Development Process

# Development of DSP Systems



# Algorithm Development

- Define inputs & outputs
- Algorithm description
  - Difference equations
  - Transfer function
- High-level DSP tools
  - MATLAB, Simulink, C/C++
- Synthesized inputs

# Algorithm Development

- Benefits of high-level tools
  - Availability of libraries, toolboxes saves development time
  - Easy to debug and modify high-level language programs
  - Input/output can be stored and analyzed
  - Easing development by using floating-point data format
  - Bit-true simulations for fixed-point DSP implementation

# Selection of DSP Processors

- DSPs are available in a wide range
- Understanding of application requirements
  - Choose processor that meets project's requirements
  - Find most cost-effective solution
- Selection criteria
  - Arithmetic format
  - Performance (MIPS, MOPS, MFLOPS, MHz)
  - Price (MIPS per dollar)
  - Power consumption (MIPS per mW)
  - On-chip peripherals

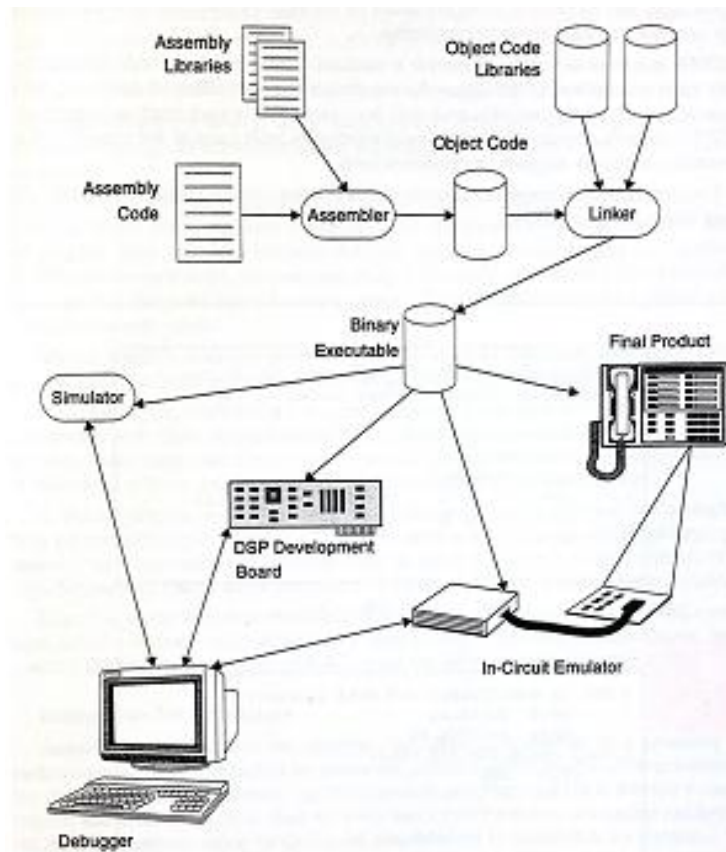
# Software Development

- Measures of good DSP software
  - Reliability
  - Maintainability
  - Extensibility
  - Efficiency
- Interdependence between hardware and software
  - DSP designer needs to understand both sides



# “Standard” DSP-Development

- Starting point: Software in C/C++



# DSP Development Tools

- Compiler / Assembler / Linker
  - For C/C++, ADA, and other HLLs
  - Performance HLL vs. hand-optimized Assembler
- Software Libraries
  - For dedicated application areas (e.g., audio/video processing)
  - Optimized for certain platforms
- Operating System
  - Manages system resources
  - Often real-time capable

# DSP Development Platforms

- (Instruction Set) Simulators
  - Simulation of program execution on host computer
  - Differences regarding accuracy, simulation speed, and completeness
- DSP Development Boards
  - DSP with peripherals as “development boards” (with connection to the developer’s workstation)
  - Evaluations can be performed, but not real target hardware
- In-circuit Emulators
  - Development support on target-hardware
  - Processor cycle, breakpoints, profiling

# Instruction Set Simulators

- Purpose
  - Executable code for desired platform
  - Execution times, memory requirements, (SW) pipelining, RTDX
- Hardware support
  - Pin connect, port connect, latency, cache, peripherals, DMA
  - Data path: GPR, FU, Cross-paths, control registers
- Performance

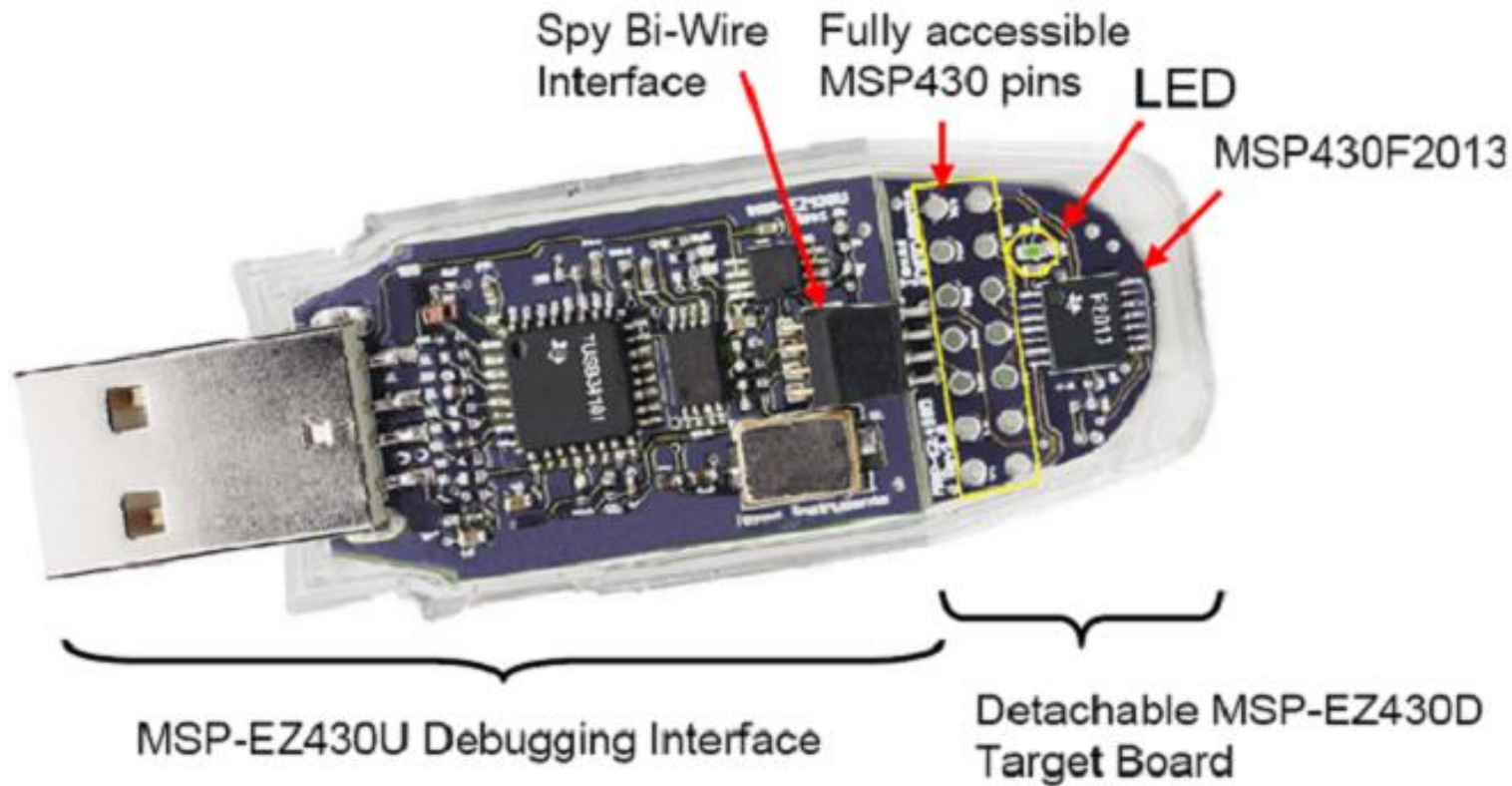
# Development Boards

- DSP Development Boards
  - DSK / EVM
  - IDE
  - Quick Start
  - Reference Design
  - DSP
  - JTAG (via PCI/LAN/USB)
  - SW Runtime Kernel, Libraries



# In-Circuit Emulators

- In-circuit Emulator (ICE) Interface



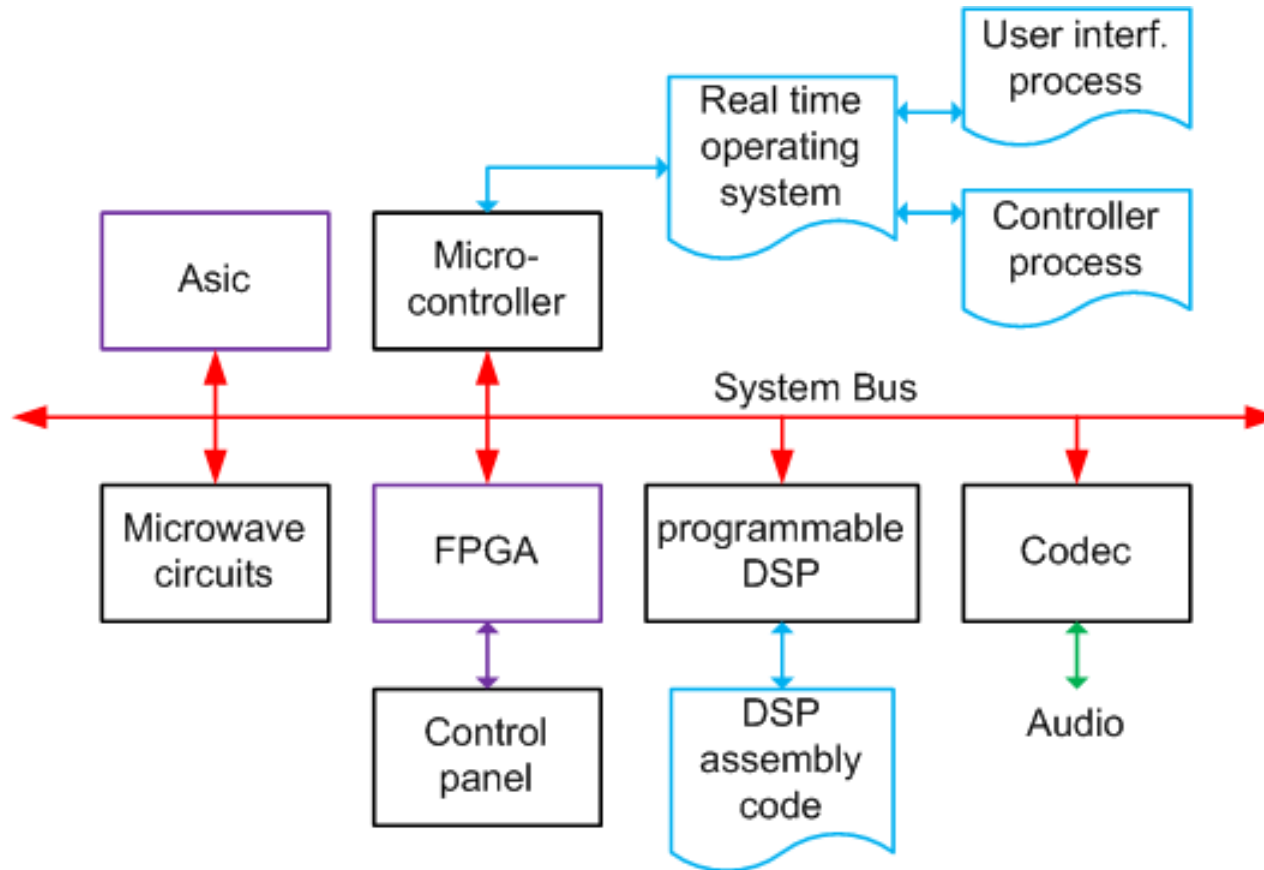
# Design Challenges

# Challenges

- **Functionality**
  - Often new “standards” (codecs, compression, etc.)
- **Heterogeneity and complexity**
  - Supported platforms
    - Special (re-configurable) Hardware, new processors
  - Specifications (language)
    - HW: VHDL, Verilog
    - SW: C, C++, MatLAB, Java
  - Complexity of target system
    - Moore’s law
    - Multiprocessor-systems, networks



# Example: Embedded DSP System



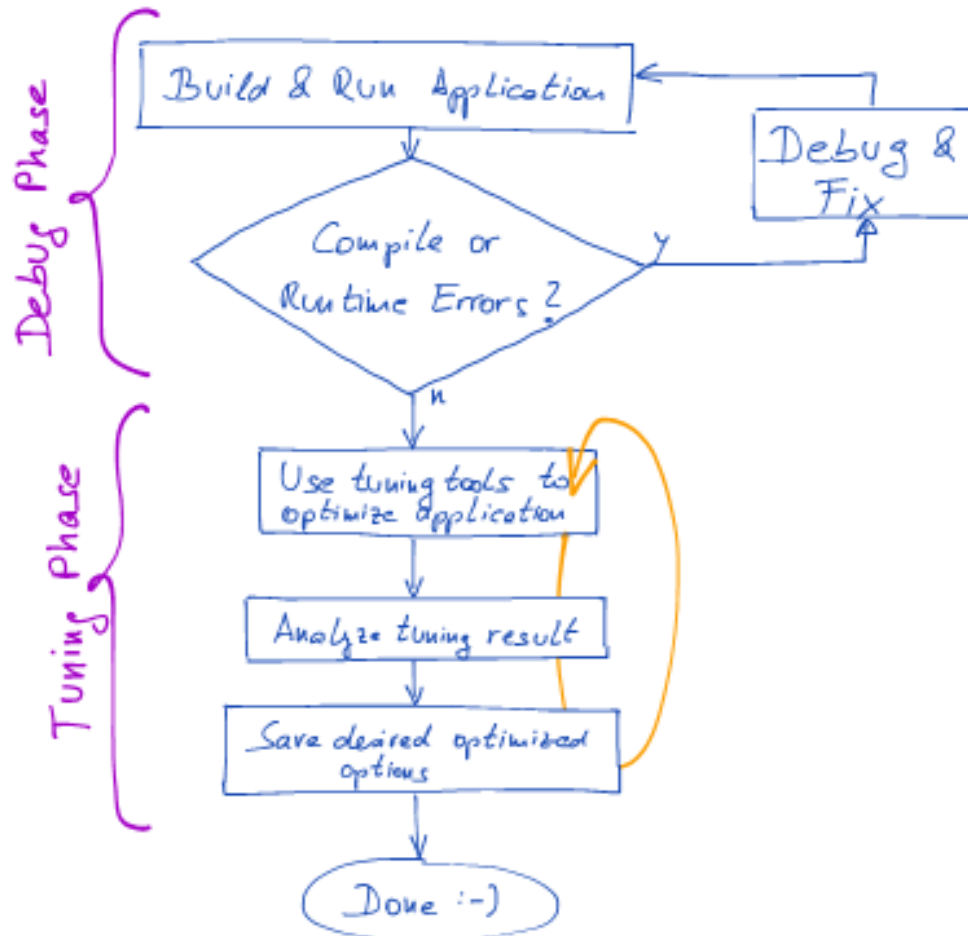
# Design Challenges

- **Functionality**
  - Correlation with specification?
  - Satisfaction of time constraints? (real-time)
- **Target System**
  - Implementation in HW or SW?
- **Application dependent Parameters**
  - Power consumption
  - Required space
  - Costs
  - . . .

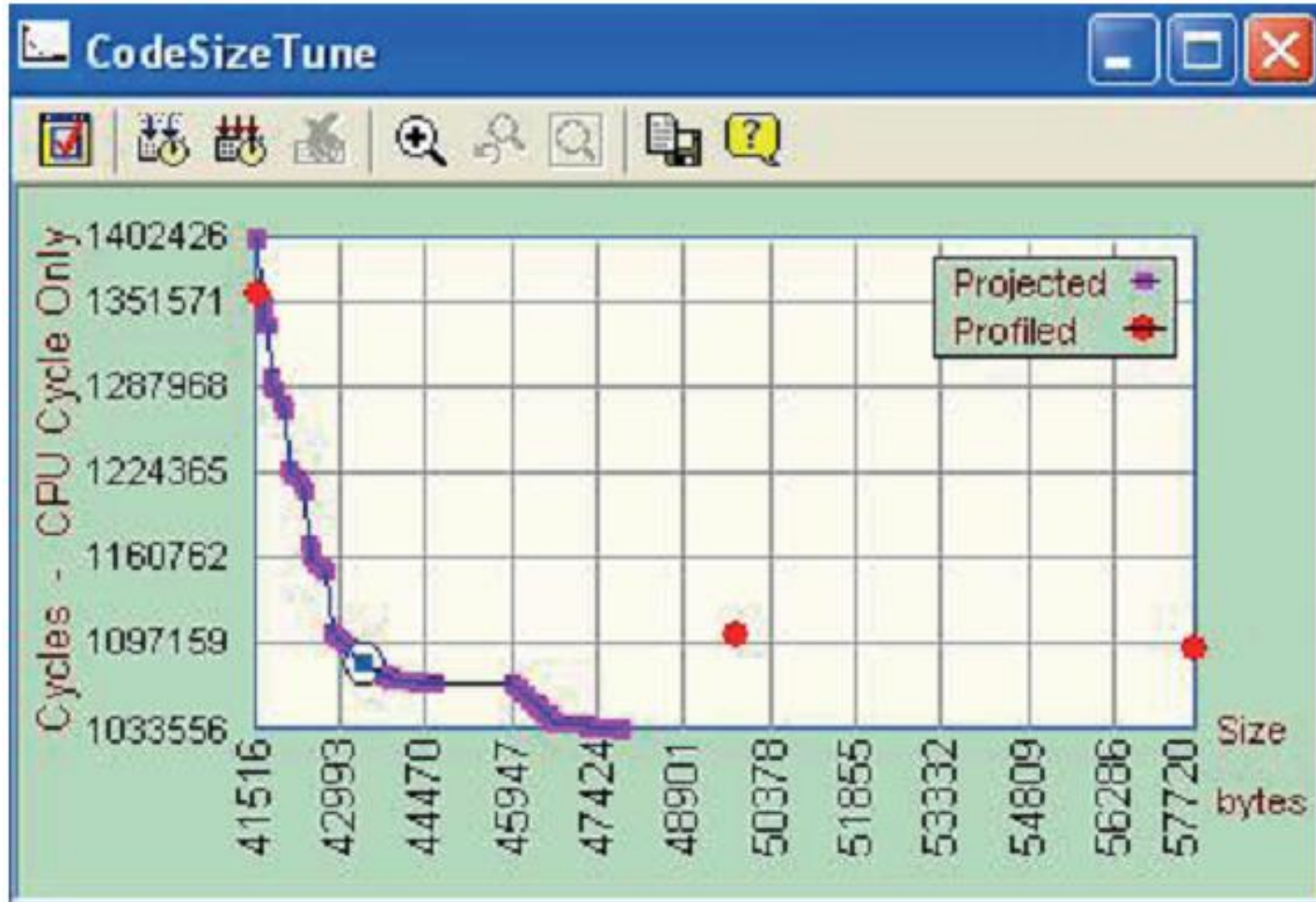
# Design Techniques

- “Specifying and synthesizing”
  - Specify requirements / functionality at different granularity (executable behaviour description)
  - Automatic synthesis of implementation (compiler, etc.)
  - *Standard for software development (?)*
- “Specifying, exploring and refining”
  - Specification => explore implementation alternatives
  - Estimate essential properties (execution time, memory, . . . )
  - Iterative refinement of specification and implementation

# Design Space Exploration



# Execution Time vs. Memory Requirements



# Parameter Tuning

C:\CCStudio\MyProjects\mp3AppNote\mp3.orf : Overrides

## Overrides

Function	Profile Collection...
unpack_sfs	Smallest
back_bf	None
back_bf0	None
bs_fill	MaximumSpeed
compare	Fastest
cvt_to_wave	Fastest
cvt_to_wave_init	None
cvt_to_wave_test	None
dummy	None

Rule

Smallest
  Fastest
  None

Apply Override

Profile Collection...	Cycle	Size	Options
<input checked="" type="radio"/> MaximumSpeed	4570	305	-o3 -oi0
<input type="radio"/> Speed	5410	233	-o3 -oi0 -ms1
<input type="radio"/> MinimumSize	5520	209	-o3 -oi0 -ms3

# Modelling of DSP Systems

# Modeling of DSP Systems

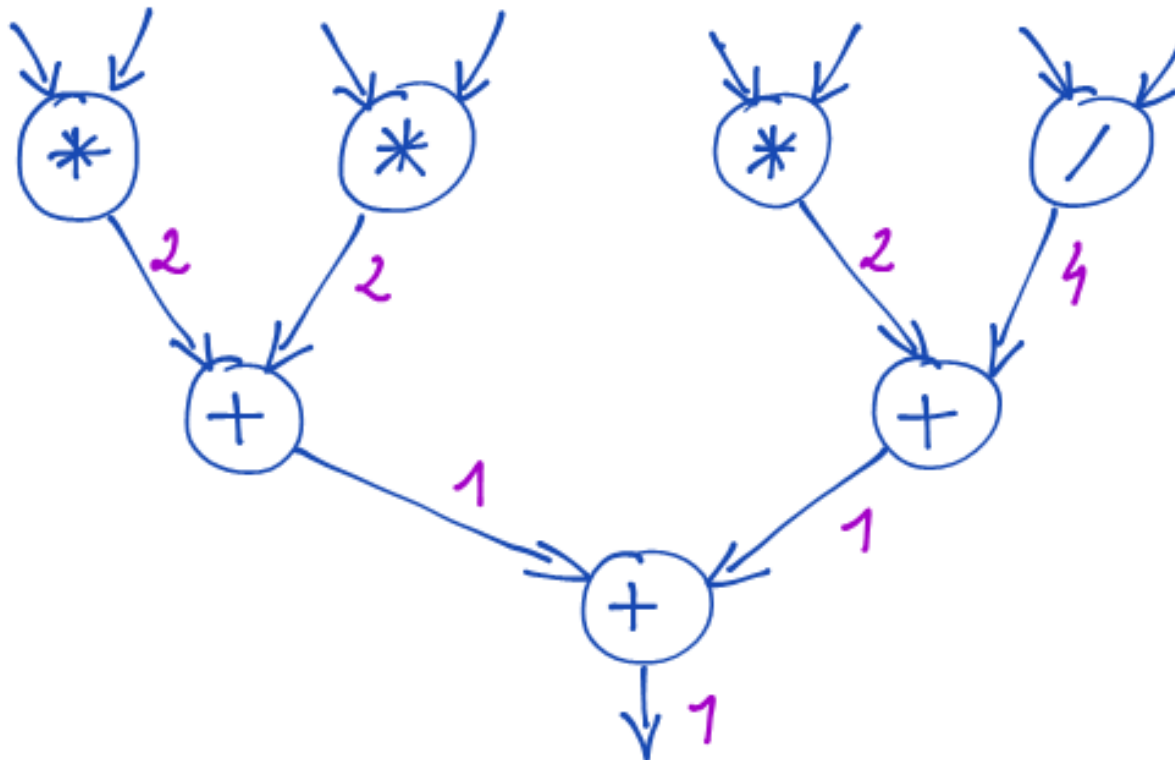
- Model
  - Formal description of a (sub-)system
  - Abstraction: description of specific characteristics, omitting unnecessary details
- Level of abstraction
  - System
  - Module / architecture
  - Block / logic
- Perspectives
  - Behaviour
  - Structure



# Dataflow Models

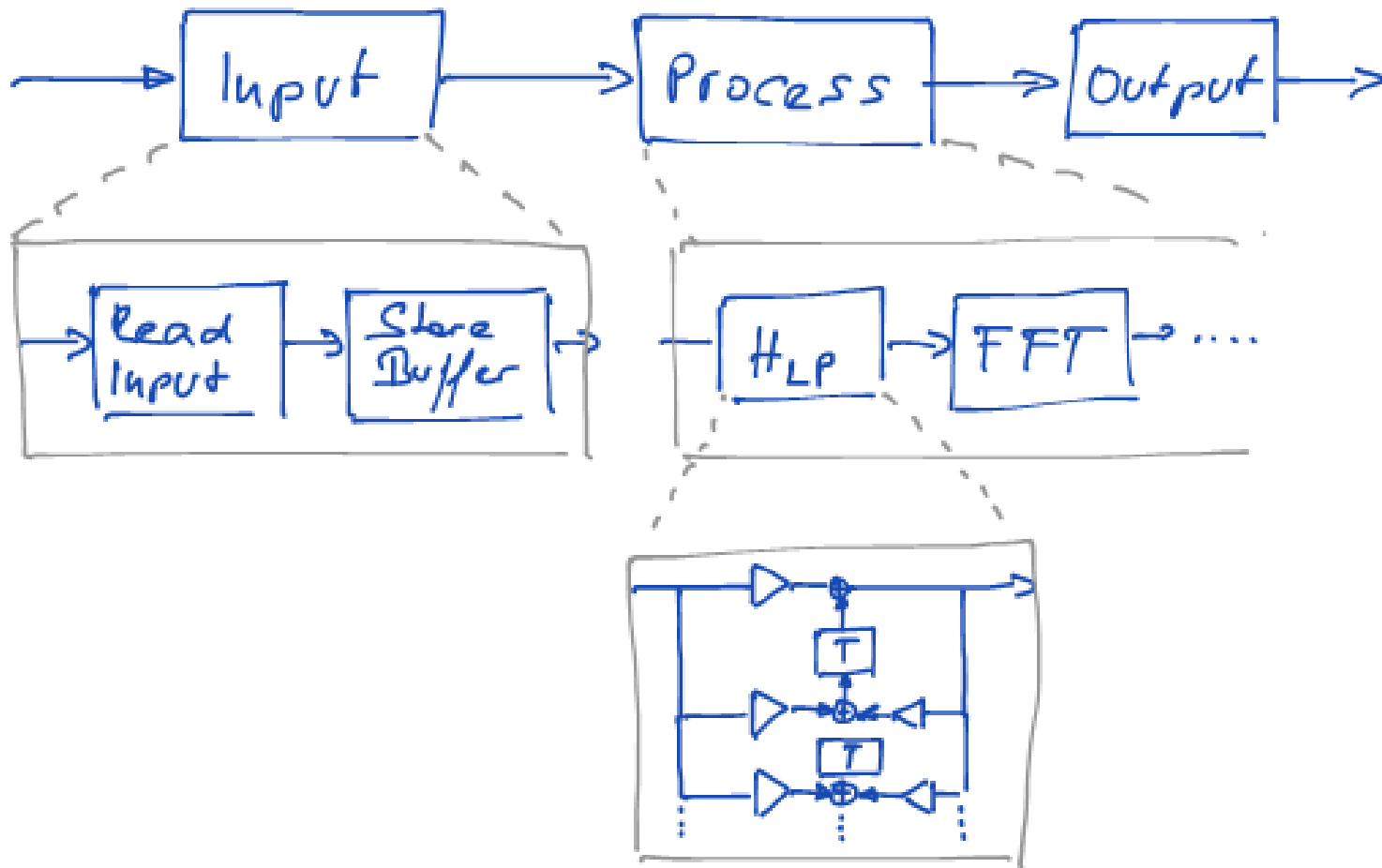
- Common description of DSP-algorithms
  - Represented as directed graph
    - Nodes: processing units
    - Edges: dataflow (in-/output) between nodes
  - “Processing” if all input-data is available
    - Firing rules*
  - Static / dynamic dataflow models, depending on “firing rules”

# Dataflow Models



# Algorithm and Architecture

# Structure of a DSP Algorithm

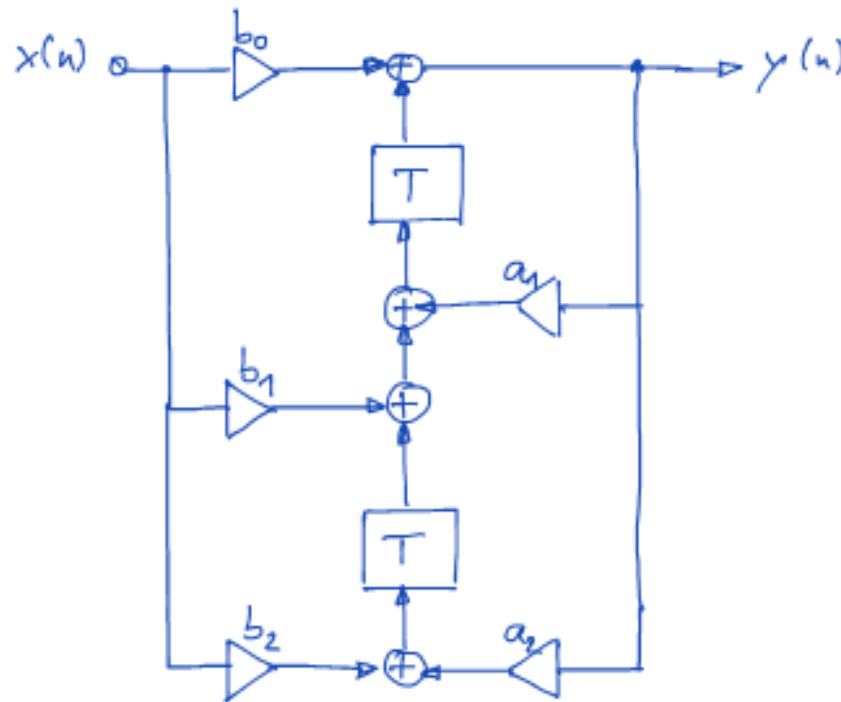


# Algorithm and Architecture

- What is the optimal hardware architecture for a given algorithm?
  - Inherent parallelism of the algorithm
- What is the minimum computation time of an algorithm for a given architecture?
  - Transform algorithm for optimal resource usage

# Example: Digital Filter

- How many Adder/Multiplier are required?
- What is the minimum execution time?



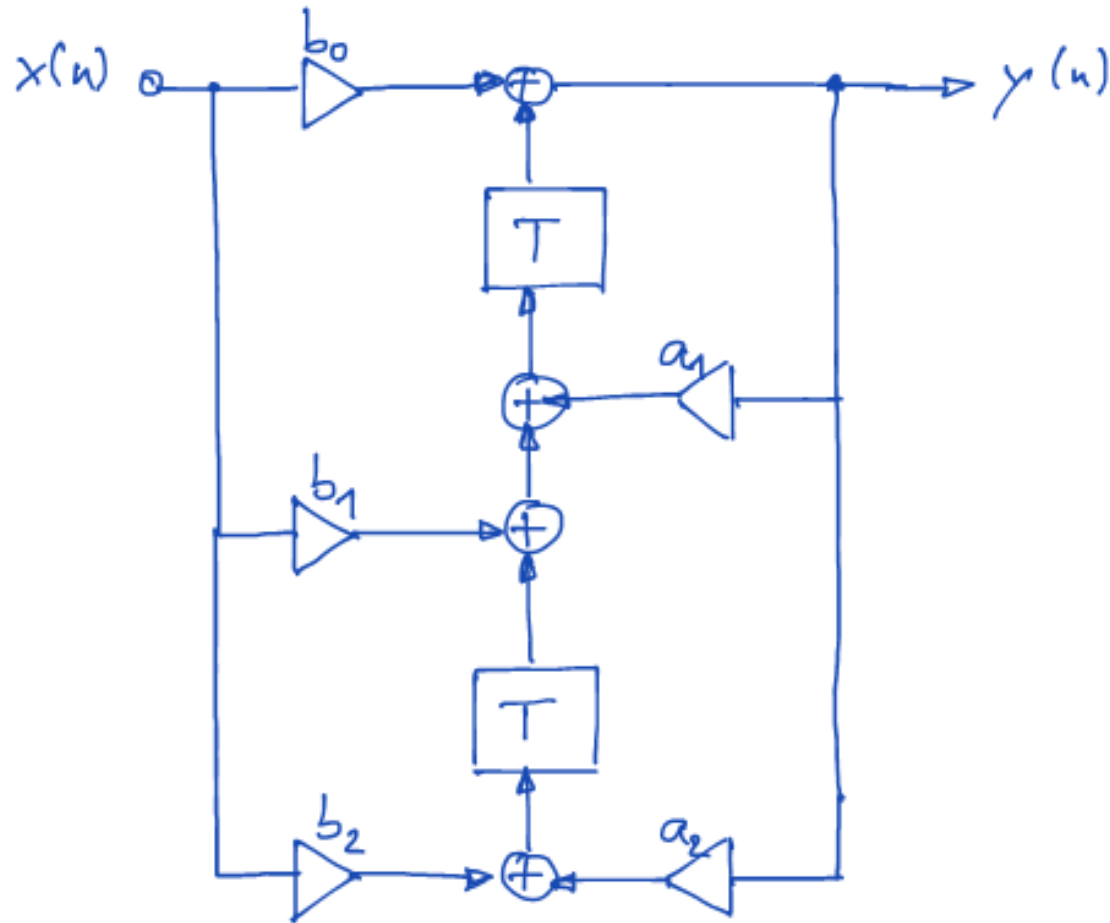
# Latency vs. Throughput

- Latency
  - Time it takes to generate an output value from the corresponding input value
- Throughput
  - Output values per second; reciprocal of the time between two outputs.
  - Increase throughput by using pipelining, for example

# Signal Flow Graph



# Signal Flow Graph (SFG) for Digital Filter



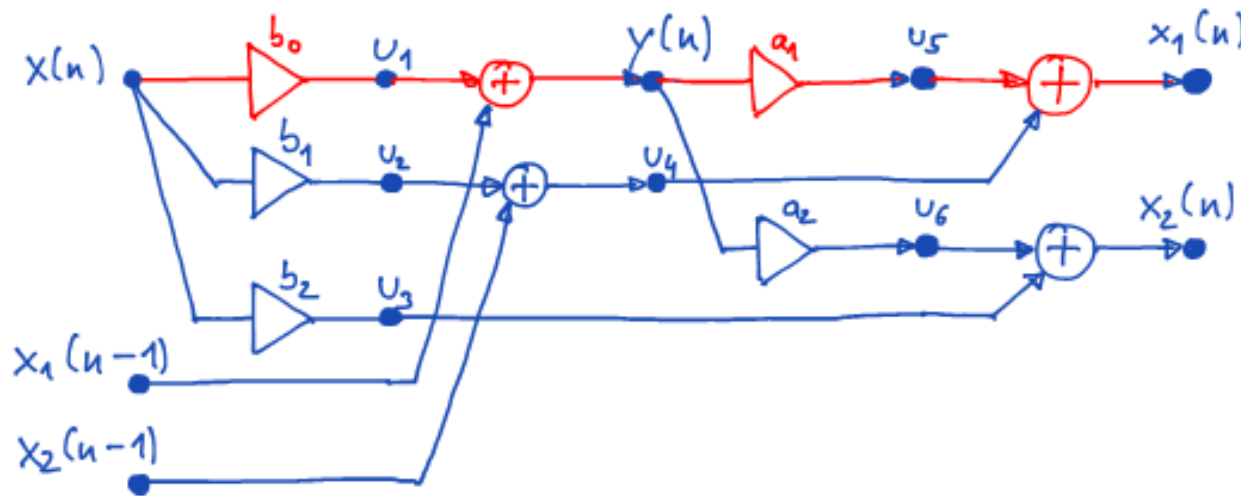
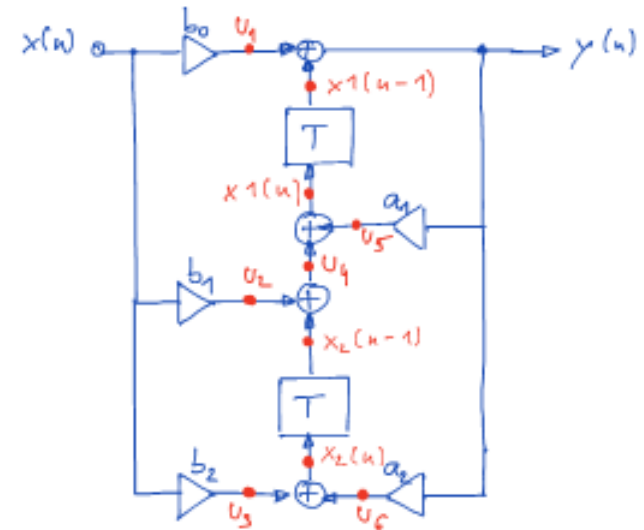
# Transform SFG in Precedence Form

- Precedence form can be easily mapped to code for a general-purpose signal processor
- Simplified view of an algorithm

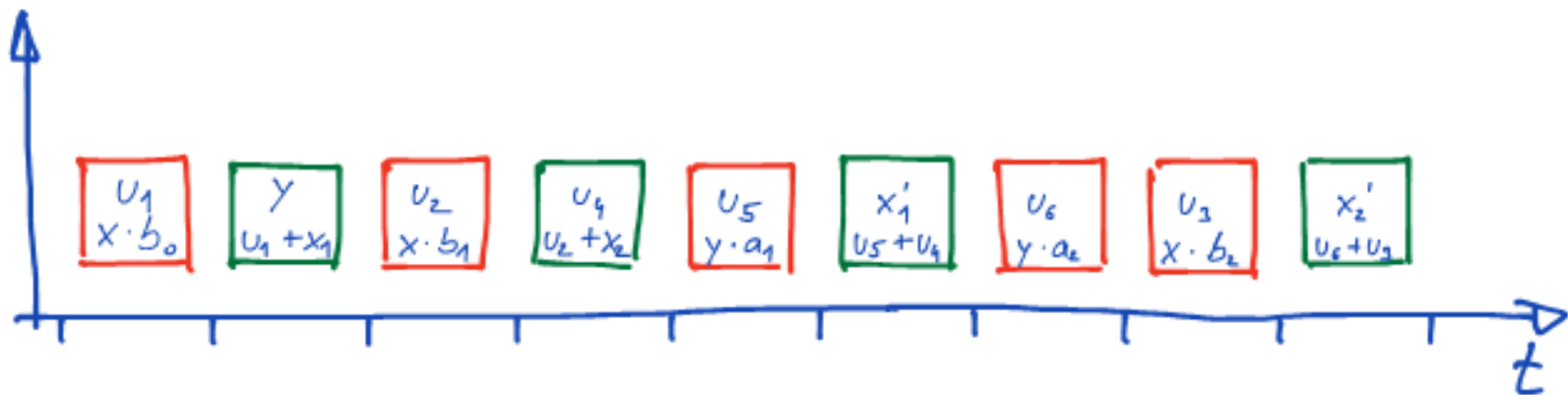
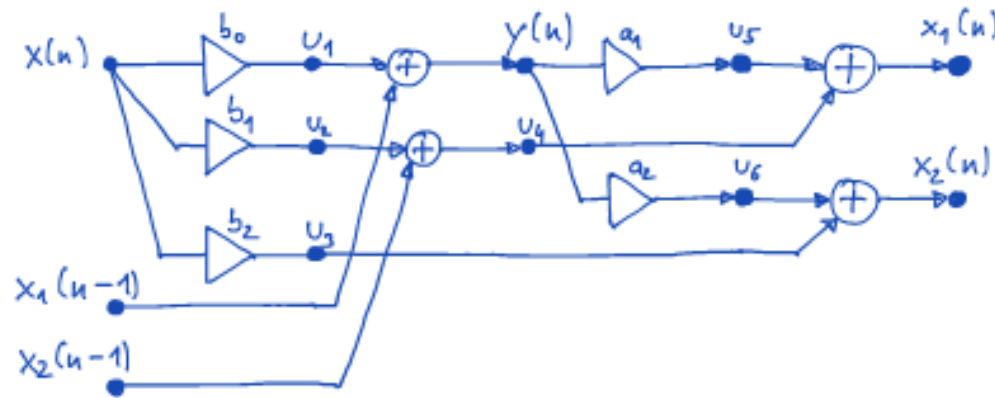
# Deriving Precedence Form of SFG

1. Collapse unnecessary nodes
2. Assign node variables: input, output, delay elements; basic operations and the results
3. Remove all edges with delay elements;  $j = 1$
4. Choose all initial nodes in the SFG and add to set  $N_j$
5. Remove edges with basic operations that are executable (i.e. for which all inputs are initial nodes)
6. Remove nodes that no longer have outgoing
7. branches
8.  $j = j + 1$
9. Repeat from step 4 until there are no initial nodes left

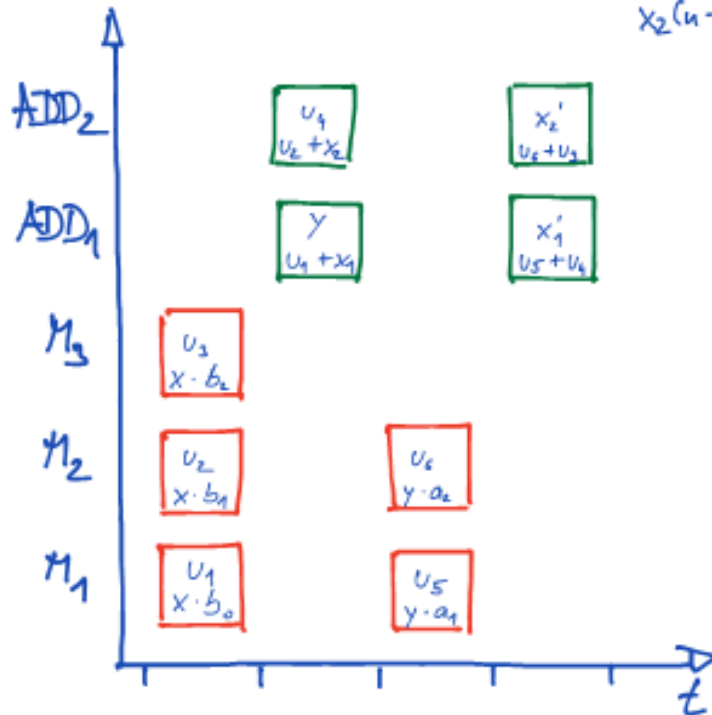
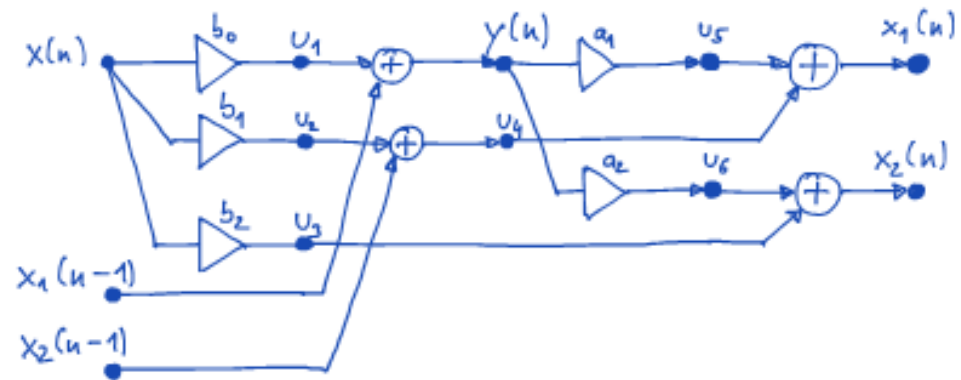
# Precedence Form of SFG



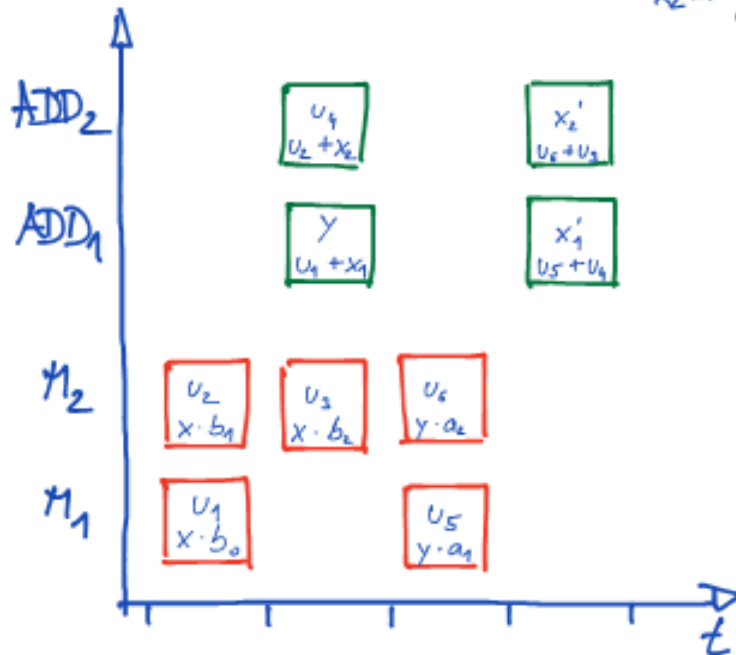
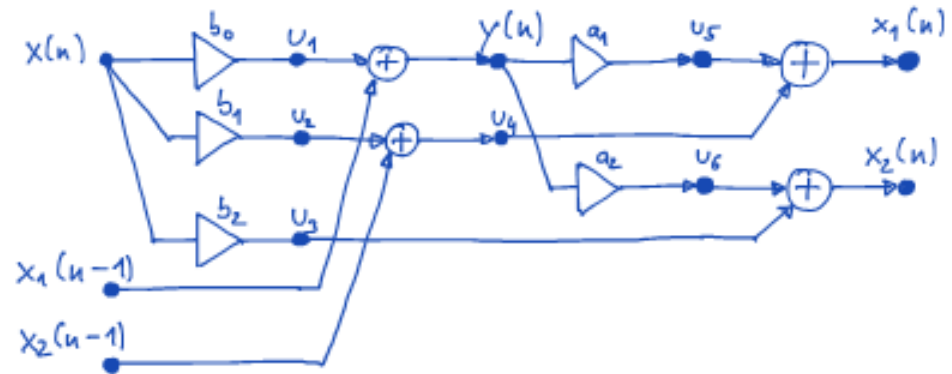
# Instruction Schedule: Scalar Processor



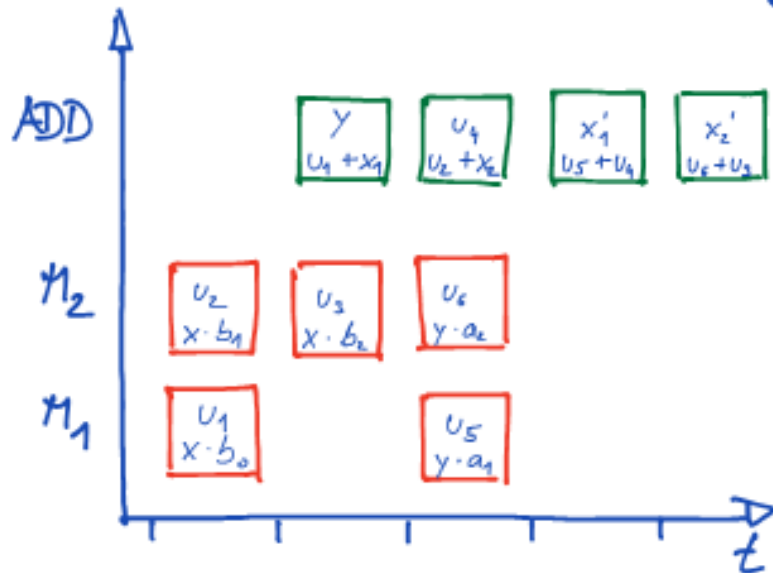
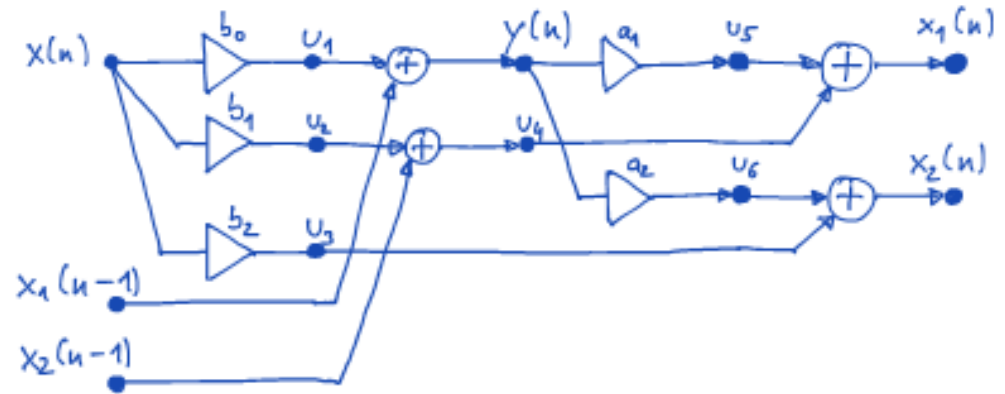
# Instruction Schedule: Shortest Execution Time



# Instruction Schedule: Optimal Schedule

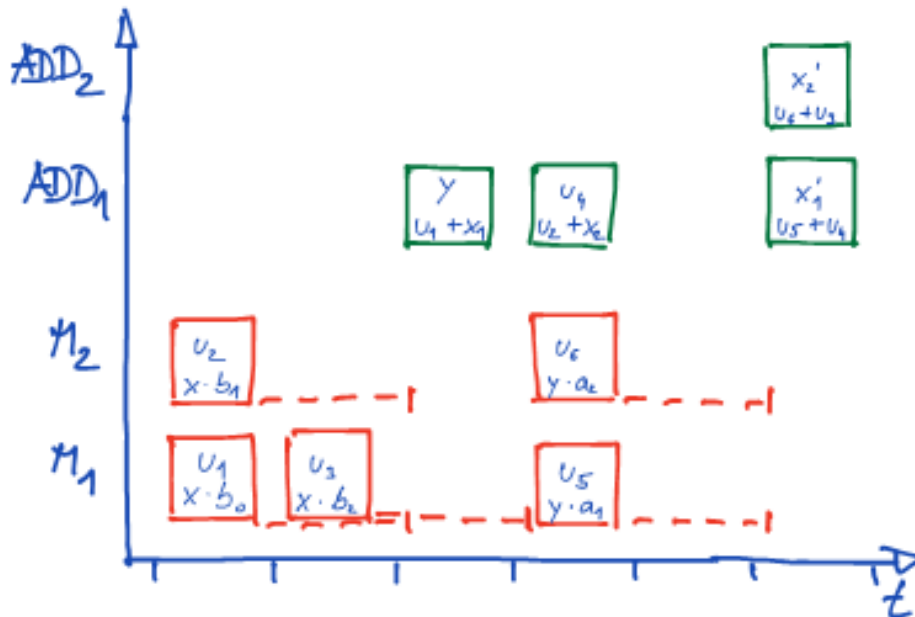
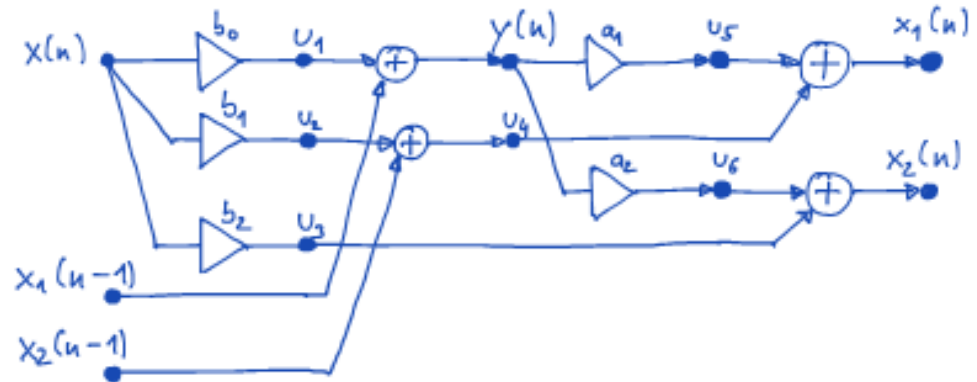


# Instruction Schedule: Suboptimal Schedule

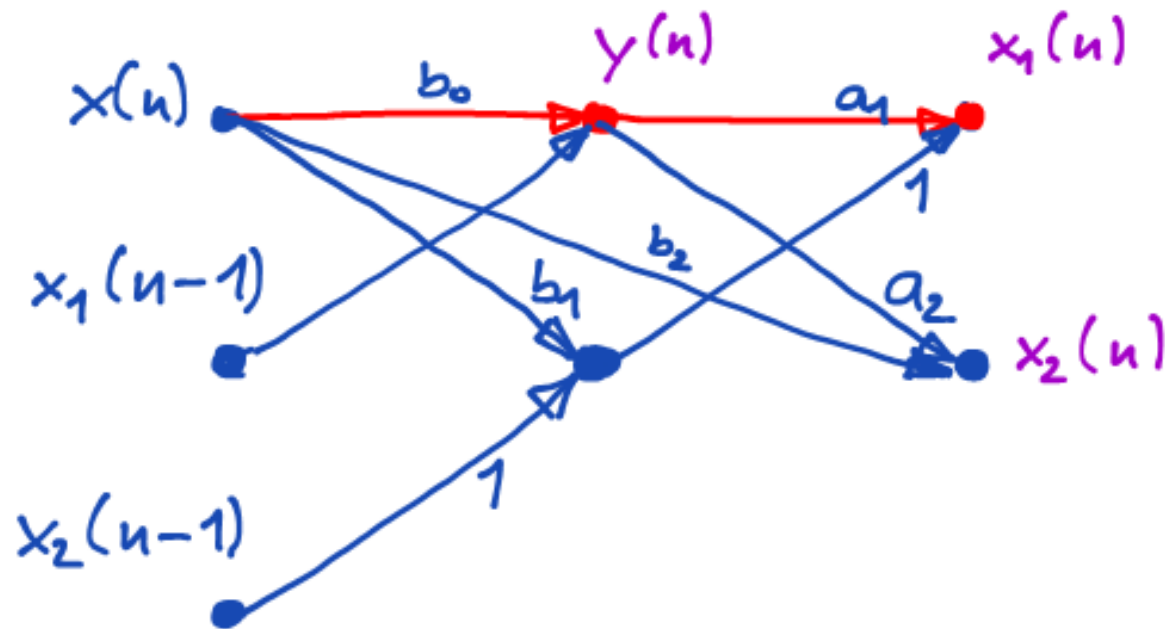




# Instruction Schedule: TI C6x DSP

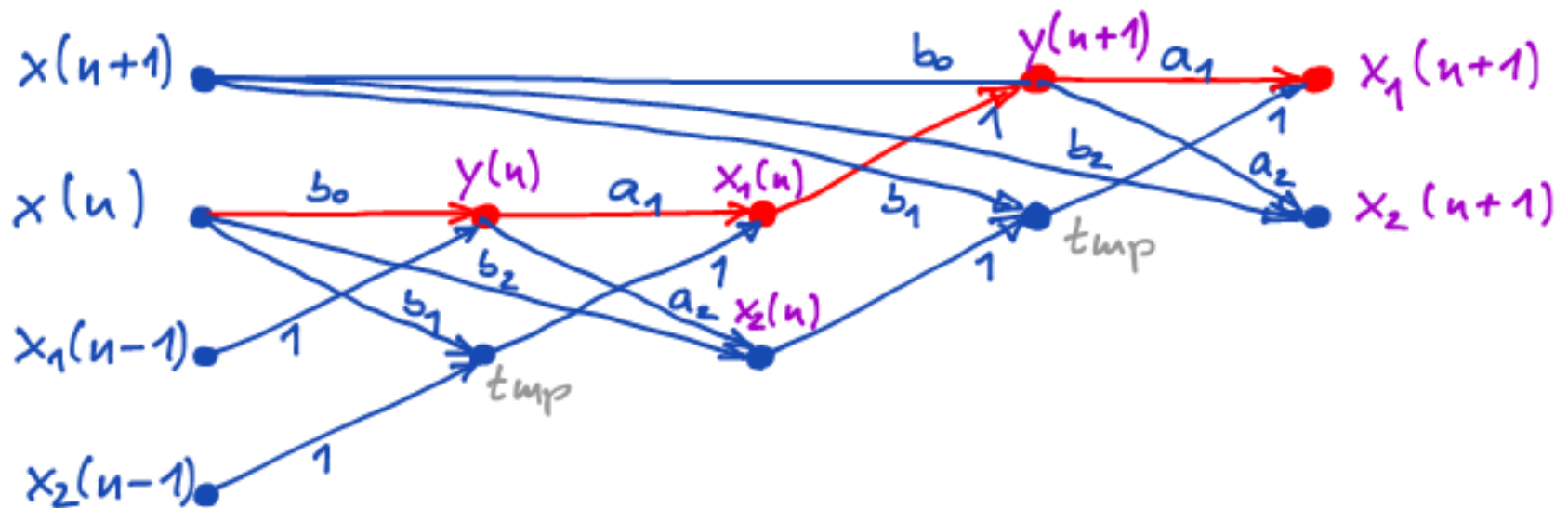


# Compact Notation of Precedence SFG

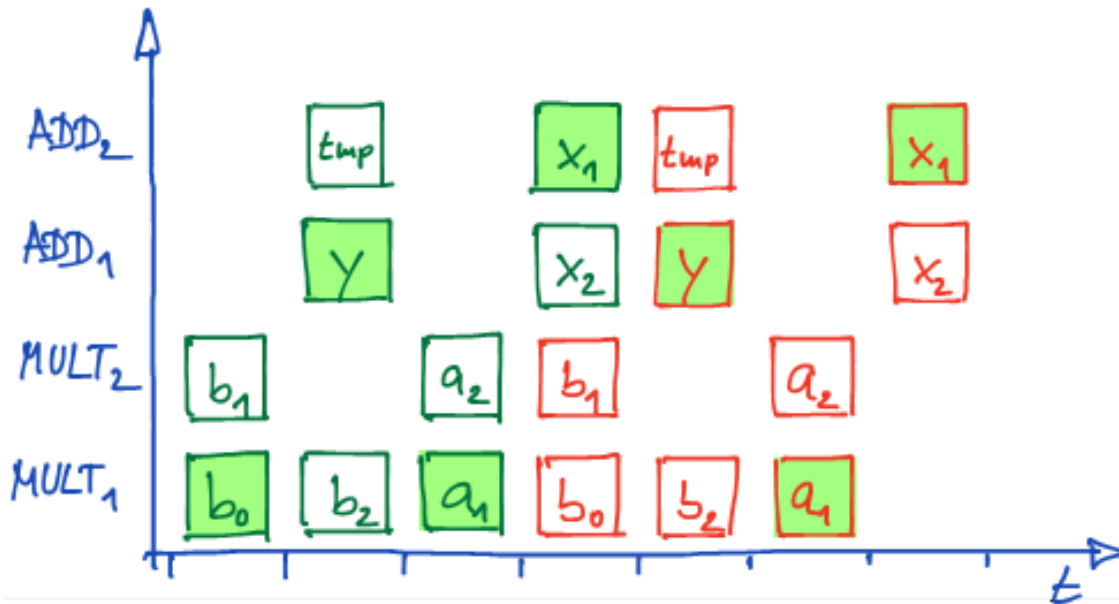
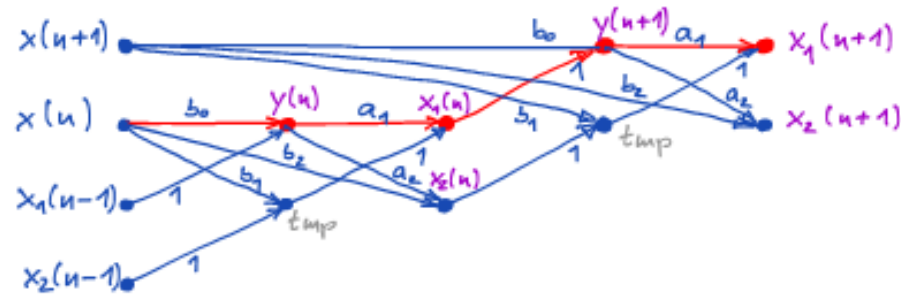


# Further Optimization: Block Data Processing

- Example: Process 2 input values together



# Schedule for 2 Input Samples

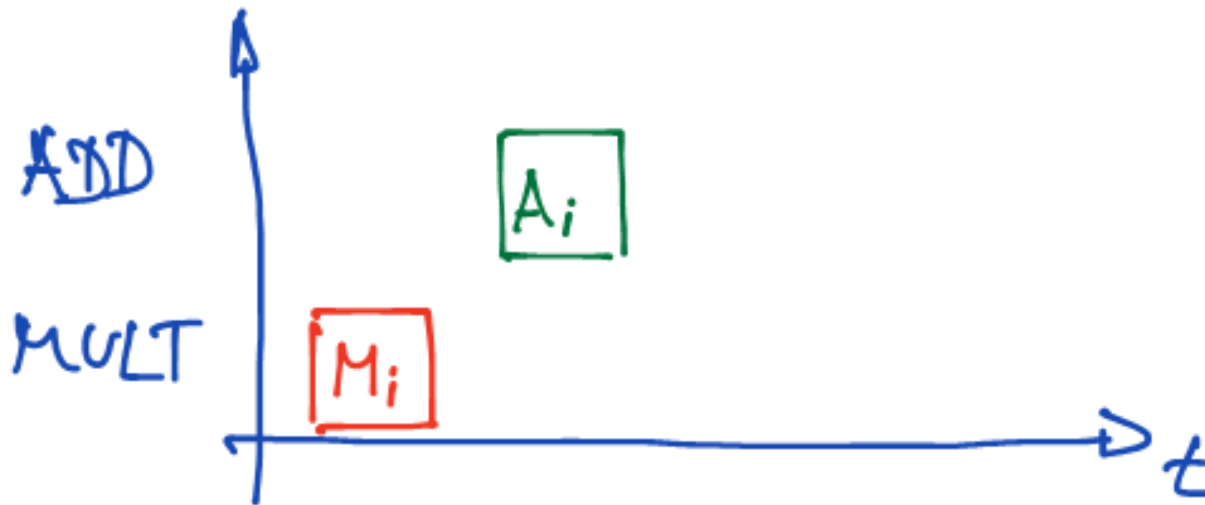
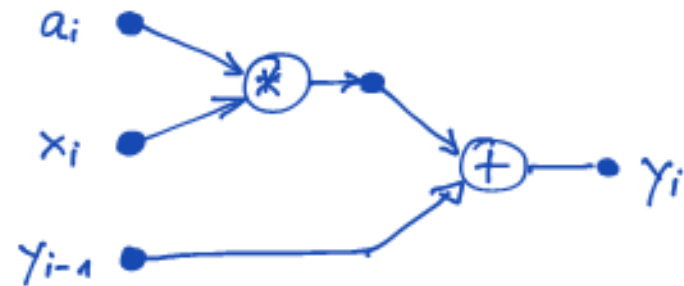


# Things we've not considered

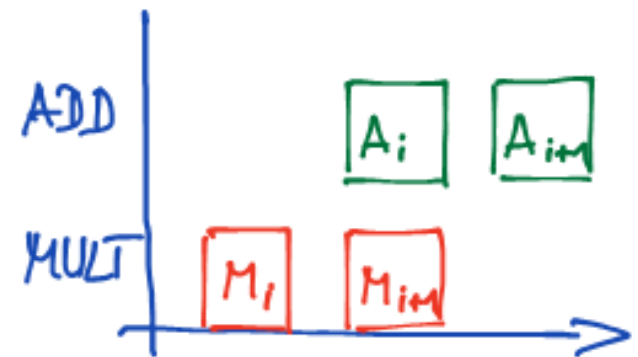
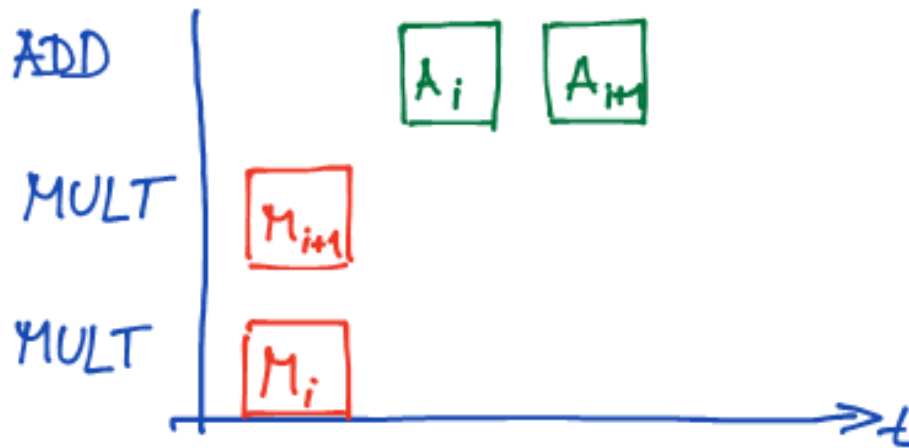
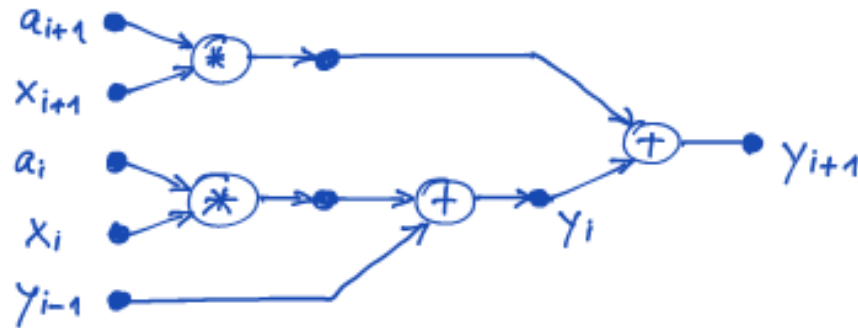
- Get input values
- Write output values
- How to get constant values ( $b_{0\dots 2}$ ,  $a_{1,2}$ )
  - Additional load instructions
  - Additional registers
- Special instructions available on DSP, e.g., MULT2, DOTP2
  - Virtually doubles the number of available units
- Limitations of the underlying DSP hardware
  - e.g, crosspaths, supported operations of functional units

# Example: Dot Product

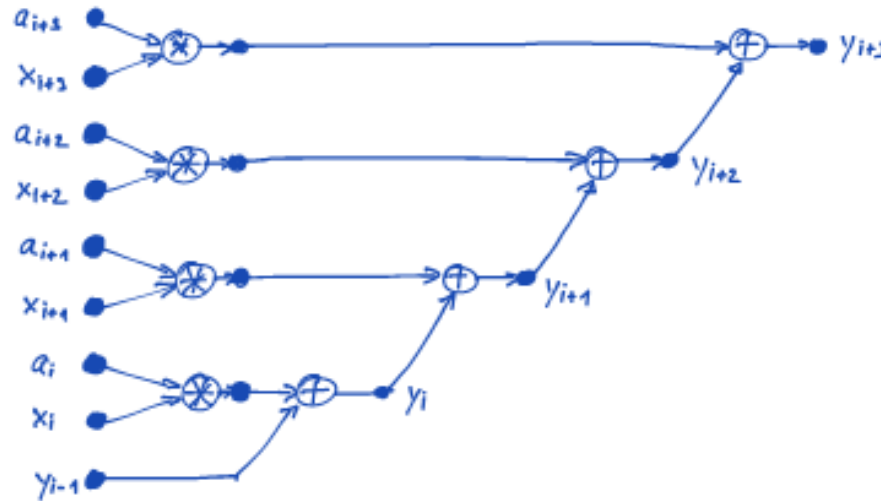
$$y_i = a_i \cdot x_i$$



# Example: Dot Product – Block Processing

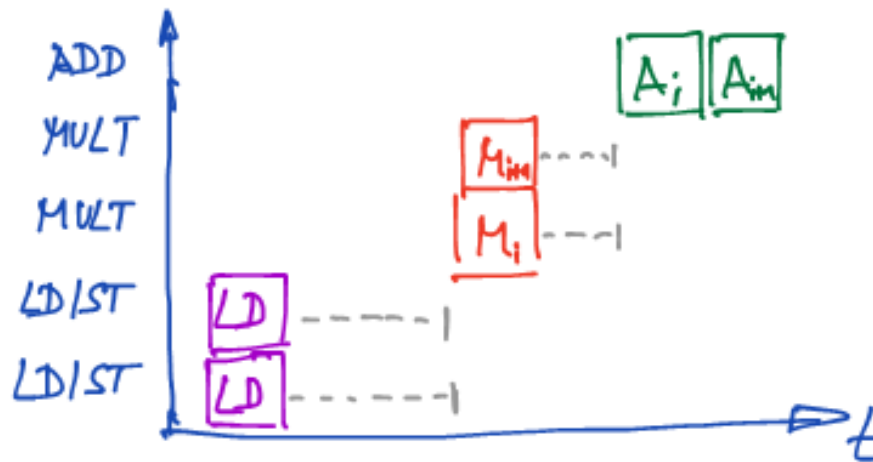
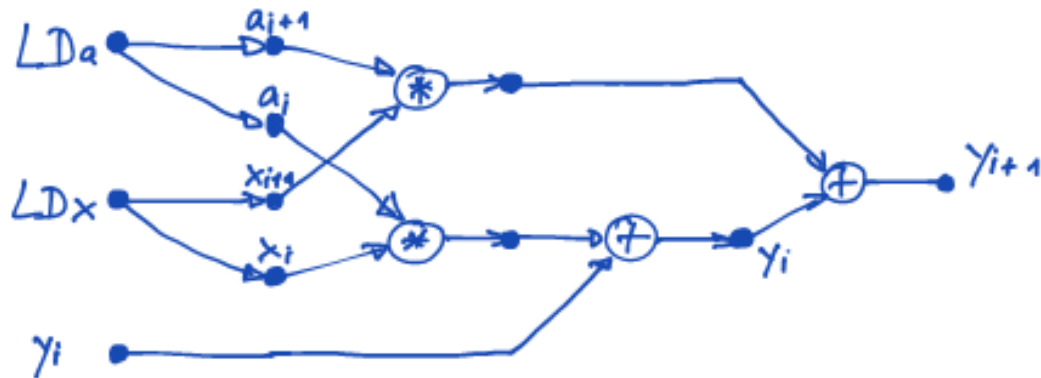


# Example: Dot Product – Block Processing





# Example: Dot Product Towards a Software Pipeline



# Software Architecture – DSP/BIOS

# No Software Architecture

- “Prototype implementation”, “programming spikes”
  - Quickly getting complex
  - Limited Maintenance and extensibility
  - “Hardware-oriented” programming (performance)

# No Software Architecture

```
Main() { /* sequential processing example */
0
  bEventFlag0 = FALSE;
  bEventFlag1 = FALSE;
  ...
  StartSystem(); /*enable interrupts */

  while(1) { /* sequential processing loop */
    if bEventFlag0 {
      bEventFlag0 = FALSE;
      ProcessEvent0();
    }
    if bEventFlag1 {
      bEventFlag1 = FALSE;
      ProcessEvent1();
    }
  }
0
}
```

```
Event_0_ISR
0
  bEventFlag0 = TRUE
0
```

```
Event_1_ISR
0
  bEventFlag1 = TRUE
0
```

# No Software Architecture

```
Main() { /* Multi-threaded example */  
    0  
}  
  
Thread_Event_0() { /* Event 0 processing thread */  
    0  
    while(1) {  
        wait for Event_0 signal  
        ProcessEvent_0  
    }  
    0  
}  
  
Thread_Event_1() { /* Event 1 processing thread */  
    0  
    while(1) {  
        wait for Event_1 signal  
        ProcessEvent_1  
    }  
    0  
}
```

```
Event_0_ISR  
0  
signal Event_0  
0
```

```
Event_1_ISR  
0  
signal Event_1  
0
```

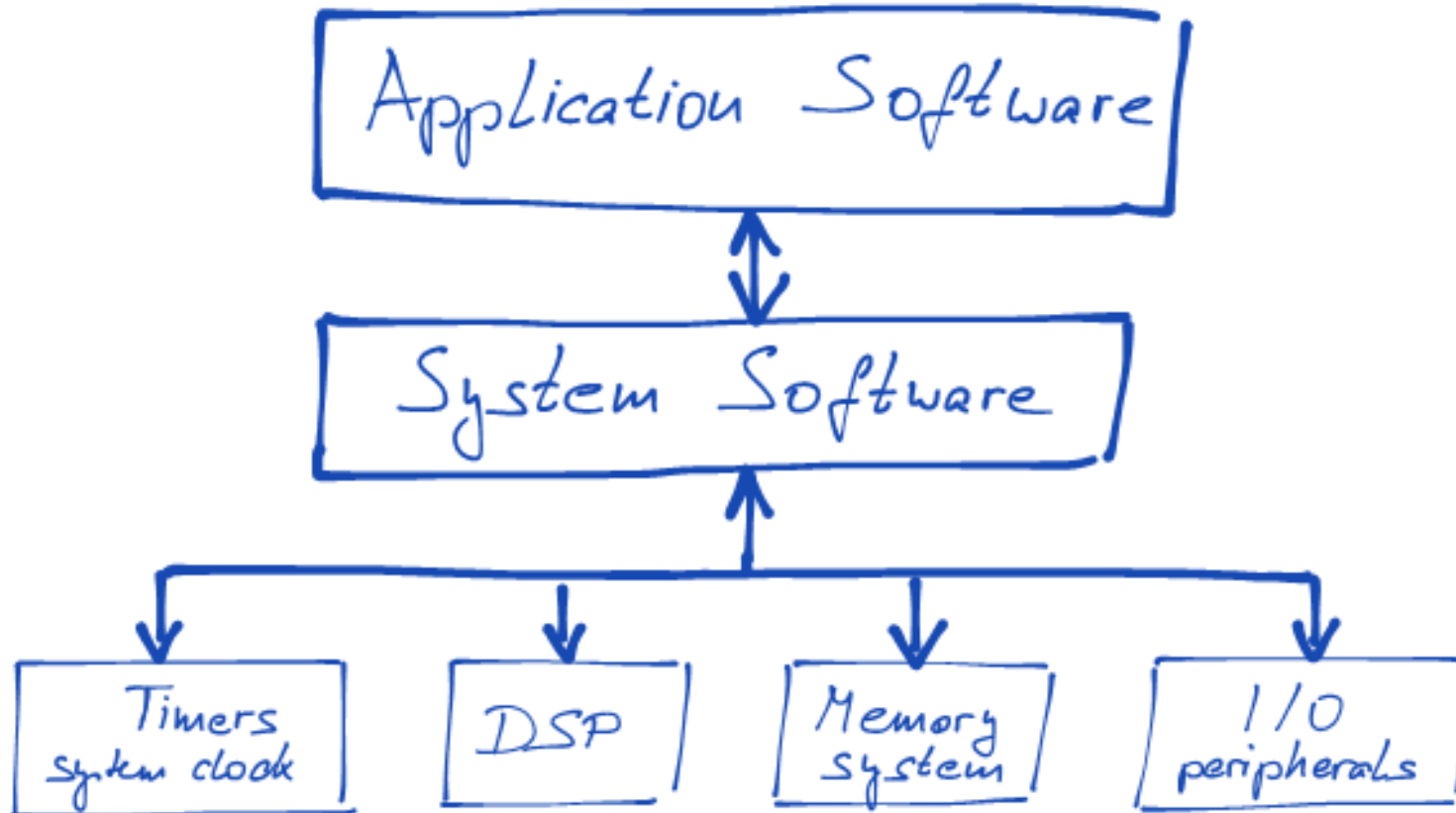
# SW Architectures for DSPs

- SW-Development “without” SW-Architecture
  - Only for small DSP systems with limited functionality
- Typical SW requirements:
  - Easy (uniform) access to hardware
  - Multithreading (with priorities)
  - Real-time abilities
  - Integration/porting of codes/programs (third-parties)
  - Simple resource management (e.g. memory)

# SW for Embedded Systems

- Additional requirements (compared to general-purpose SW)
  - Significant hardware differences, configurations, fixed-point/floating-point, memory hierarchy, . . .
  - Cost efficient (memory, processing power, . . .)
  - Time-to-market, re-use of SW modules
  - Reliability, QoS
  - Real-time constraints
- Methods, tools, etc. for efficient SW-Development
  - Operating system (for DSP, embedded system)
  - Standard for algorithms
  - Framework / SW-Architecture

# SW for Embedded Systems



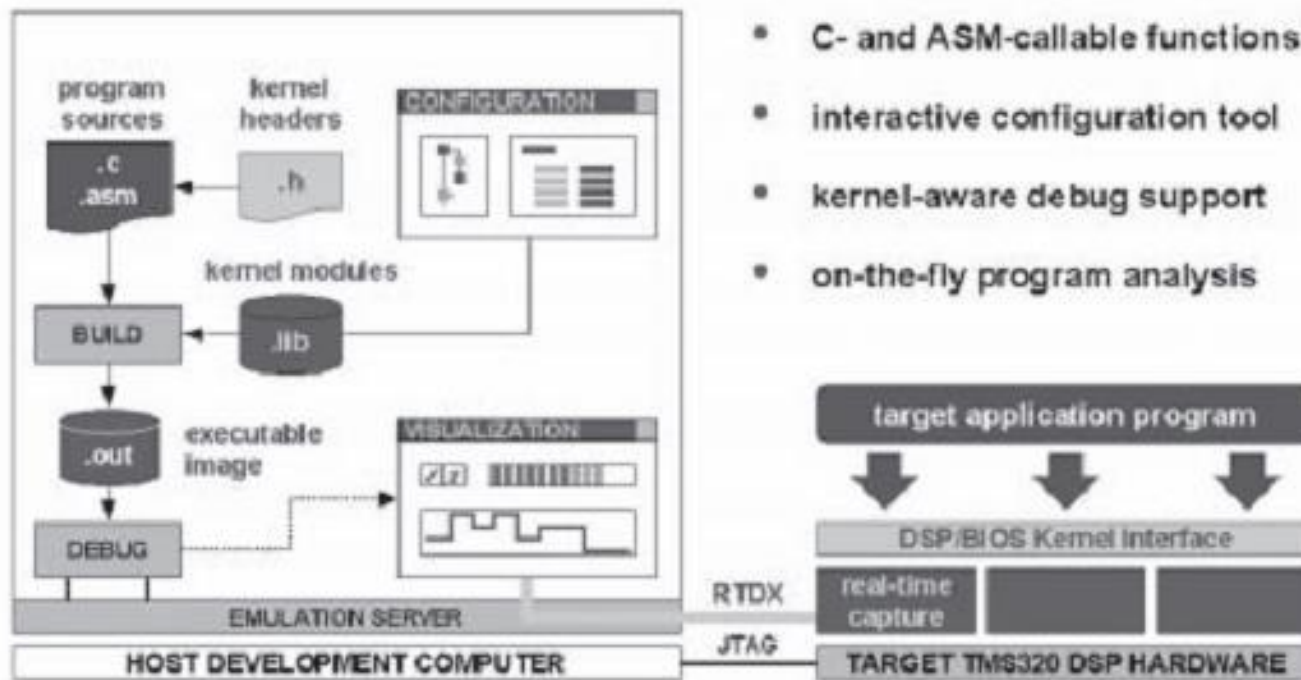


# Operating System (Core)

- Important Features
  - Hardware abstraction (interface)
  - Thread/task management
  - Thread/task communication
  - Thread/task synchronization
  - Analysis/monitoring
  
- (Commercial) operating systems for DSPs
  - VxWorks (POSIX-compatible, networking)
  - DSP/BIOS
  - WinCE
  - . . .

# DSP/BIOS (TI)

- OS Kernel integrated in CCS IDE
  - Threads with different priorities



# DSP/BIOS (TI)

- OS Kernel integrated in CCS IDE
  - Threads with different priorities

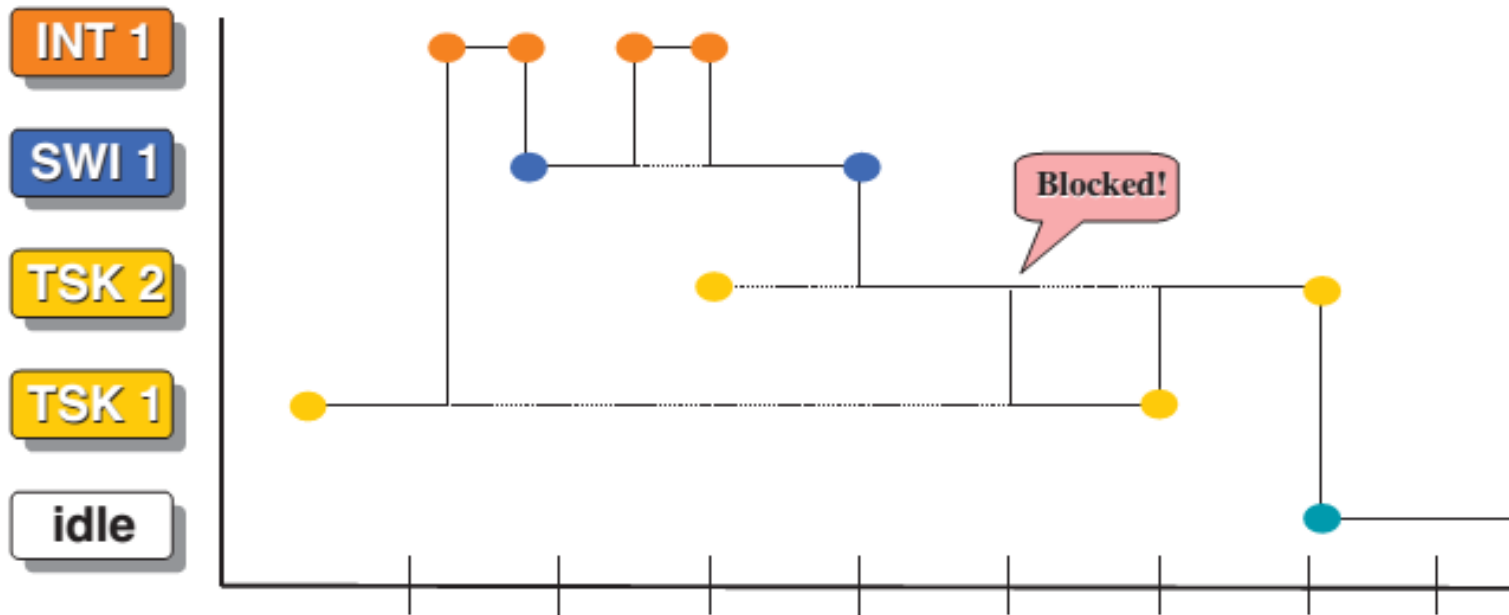
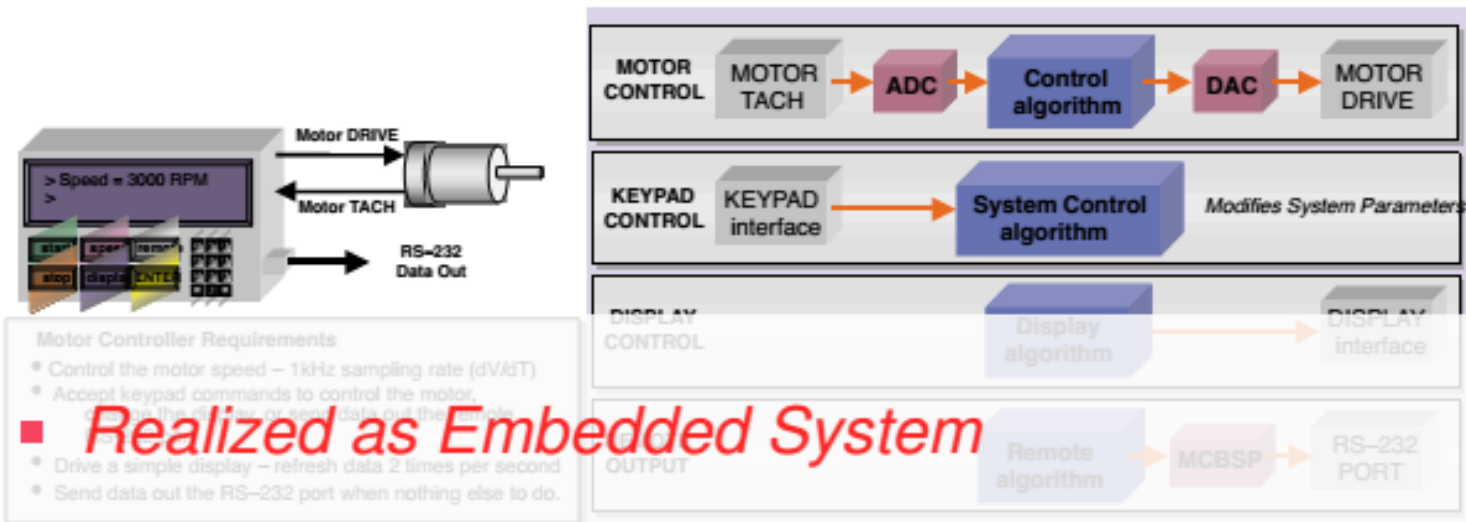


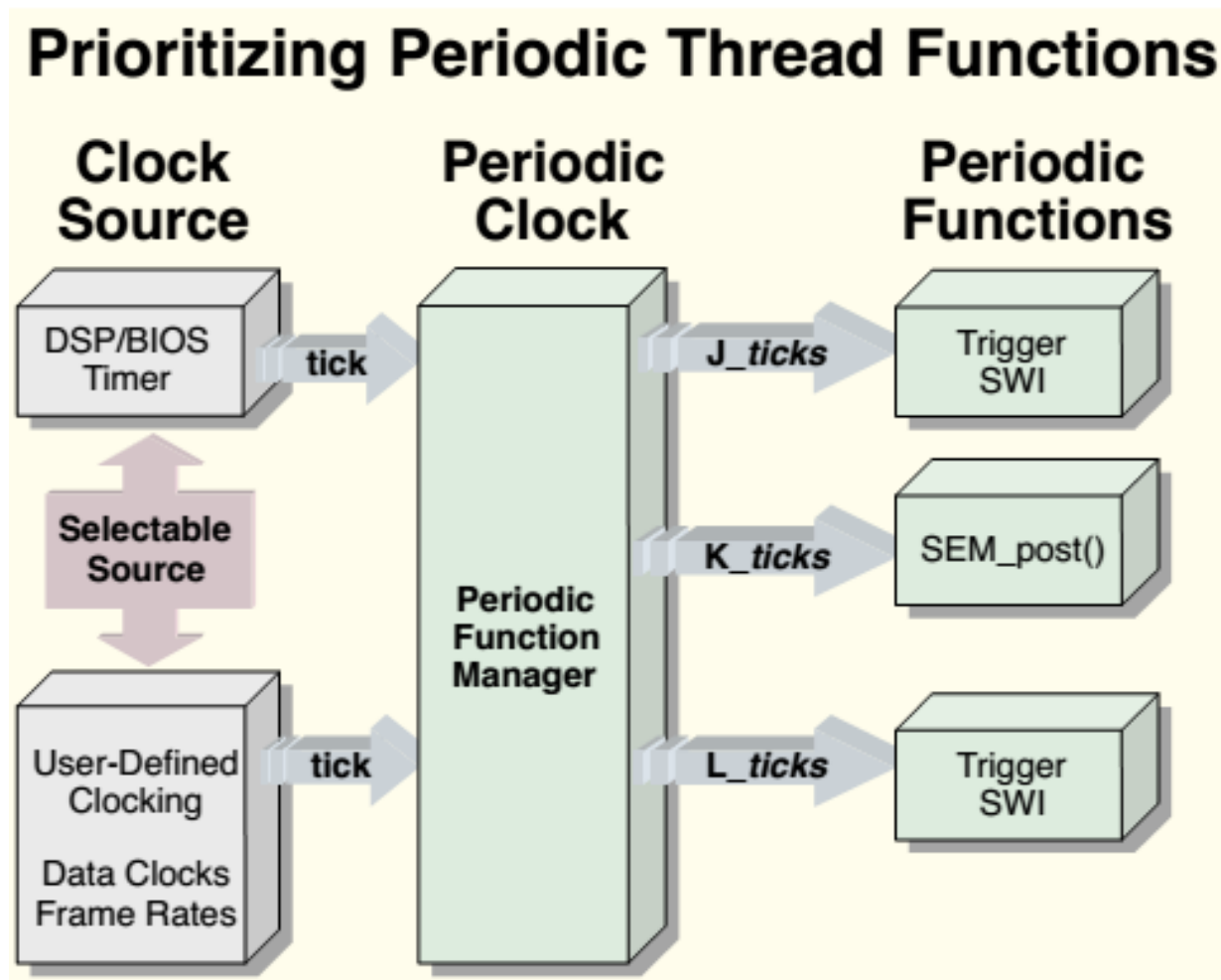
Figure 6. DSP/BIOS II Prioritized Thread Execution Model

# Example: Motor Control

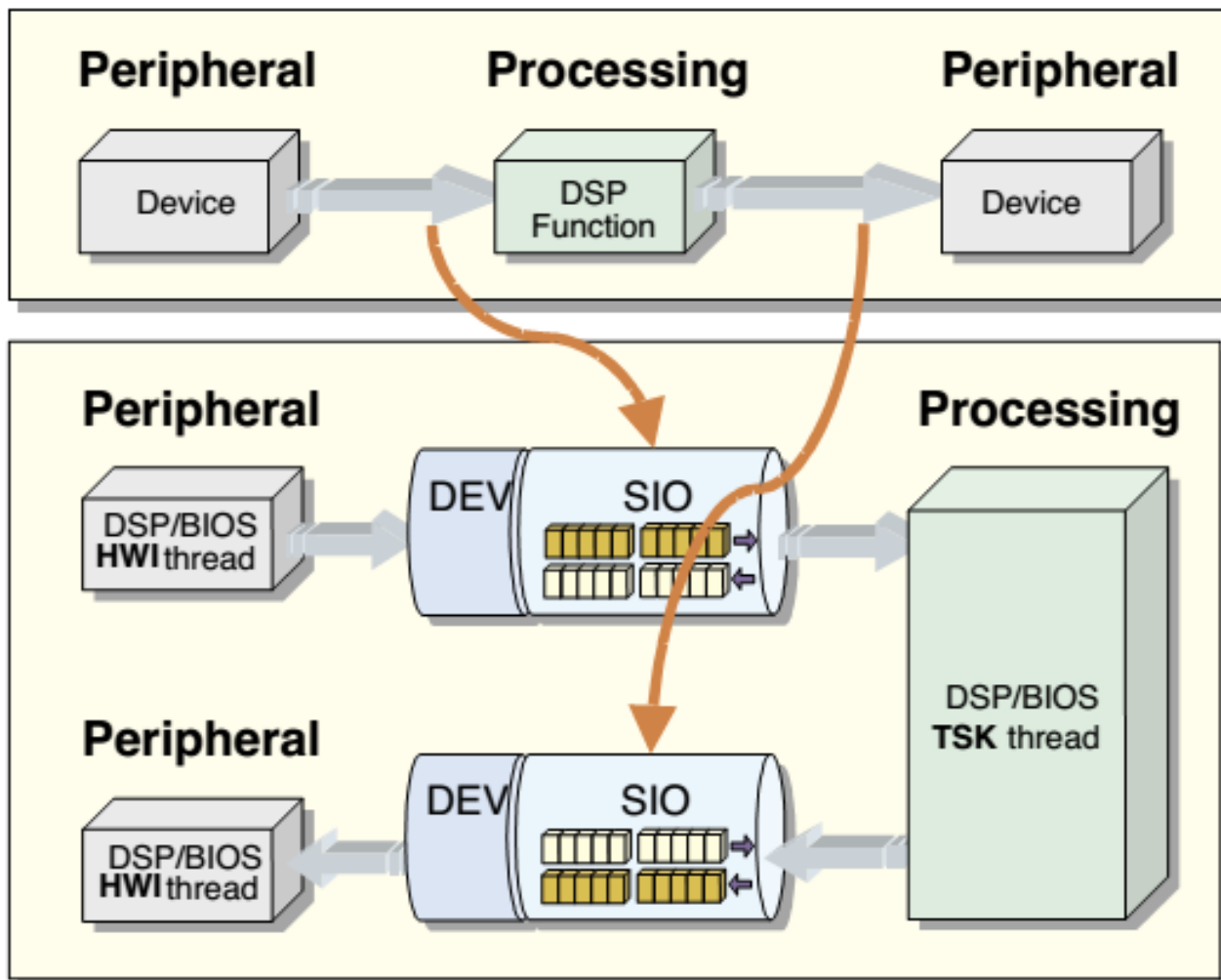
- Individual DSP carries out several functions
  - Motor control
  - Keyboard entries
  - Display
  - Data transfer



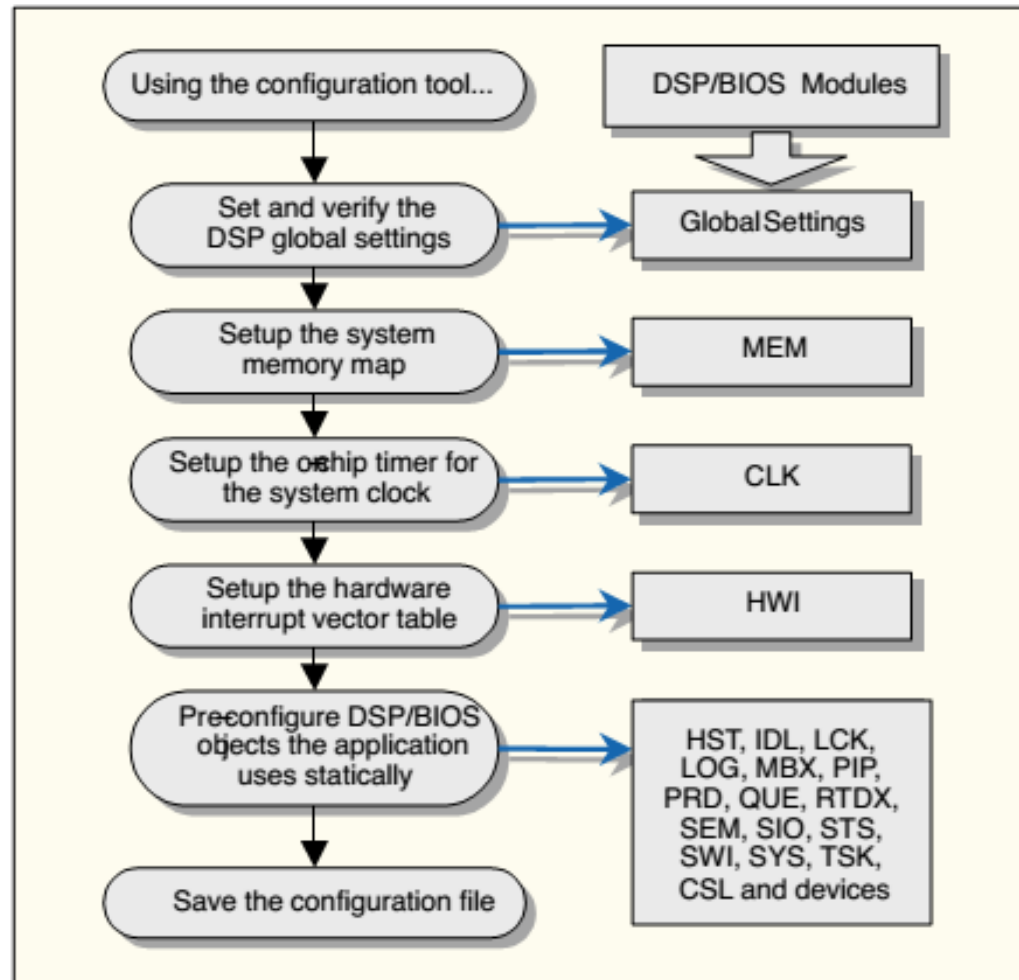
# Periodic function manager



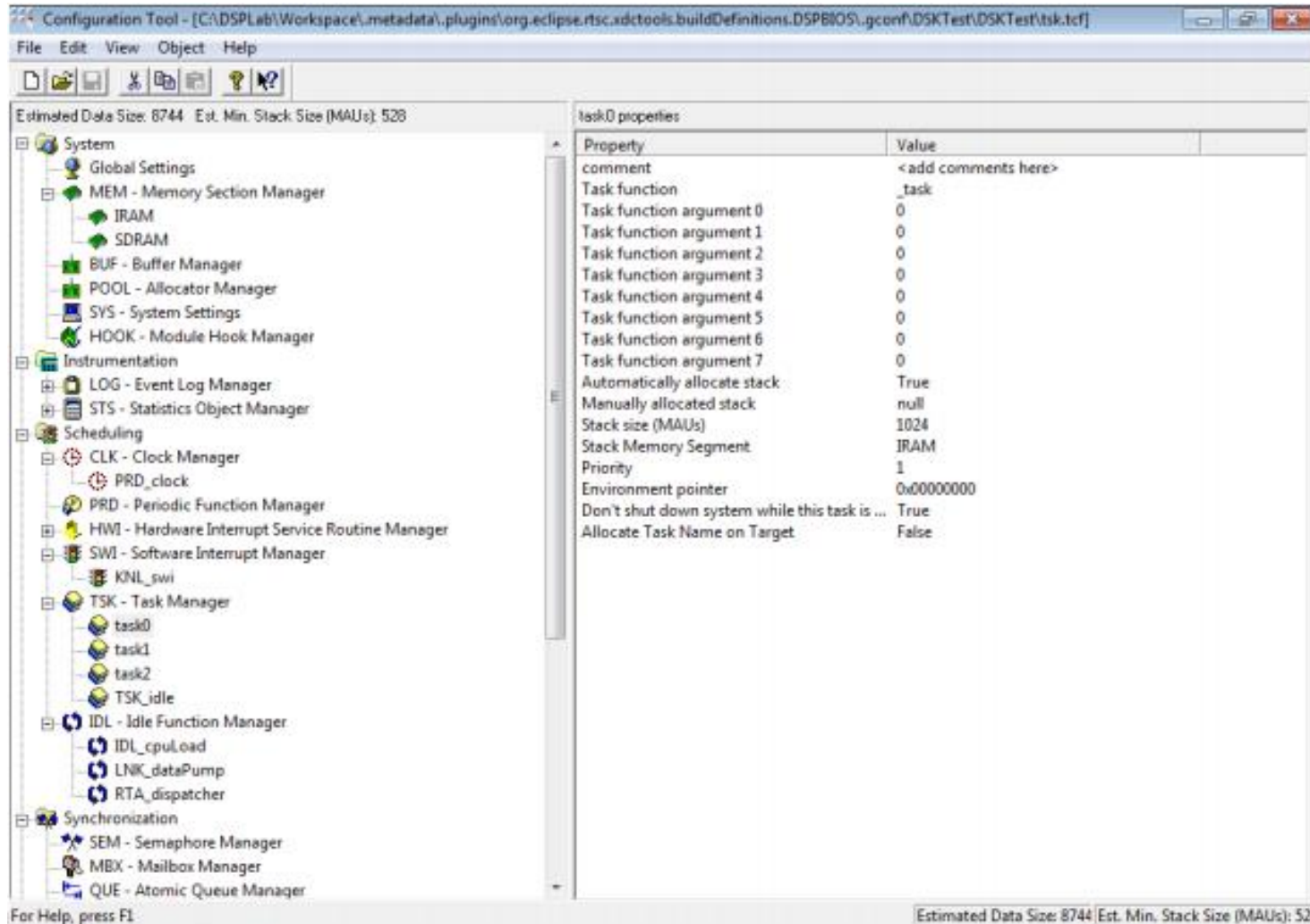
# Passing Data



# DSP/BIOS Configuration



# DSP/BIOS Configuration



Configuration Tool - [C:\DSPLab\Workspace\metadata\plugins\org.eclipse.rtc.xdctools.buildDefinitions.DSPBIOS\gconf\DSKTest\DSKTest\tsk.tcf]

File Edit View Object Help

Estimated Data Size: 8744 Est. Min. Stack Size (MAUs): 528

task0 properties

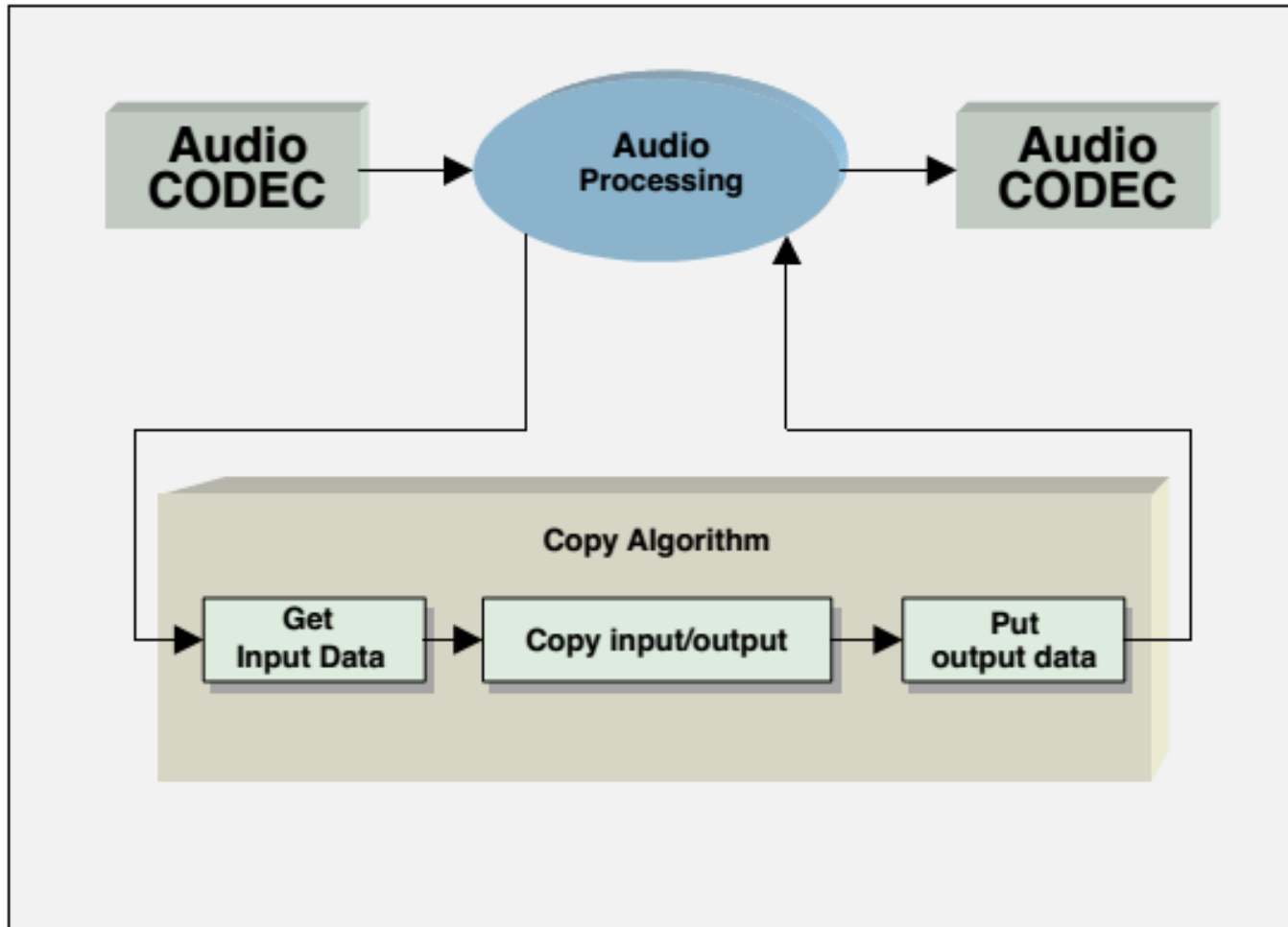
Property	Value
comment	<add comments here>
Task function	_task
Task function argument 0	0
Task function argument 1	0
Task function argument 2	0
Task function argument 3	0
Task function argument 4	0
Task function argument 5	0
Task function argument 6	0
Task function argument 7	0
Automatically allocate stack	True
Manually allocated stack	null
Stack size (MAUs)	1024
Stack Memory Segment	IRAM
Priority	1
Environment pointer	0x00000000
Don't shut down system while this task is ...	True
Allocate Task Name on Target	False

For Help, press F1

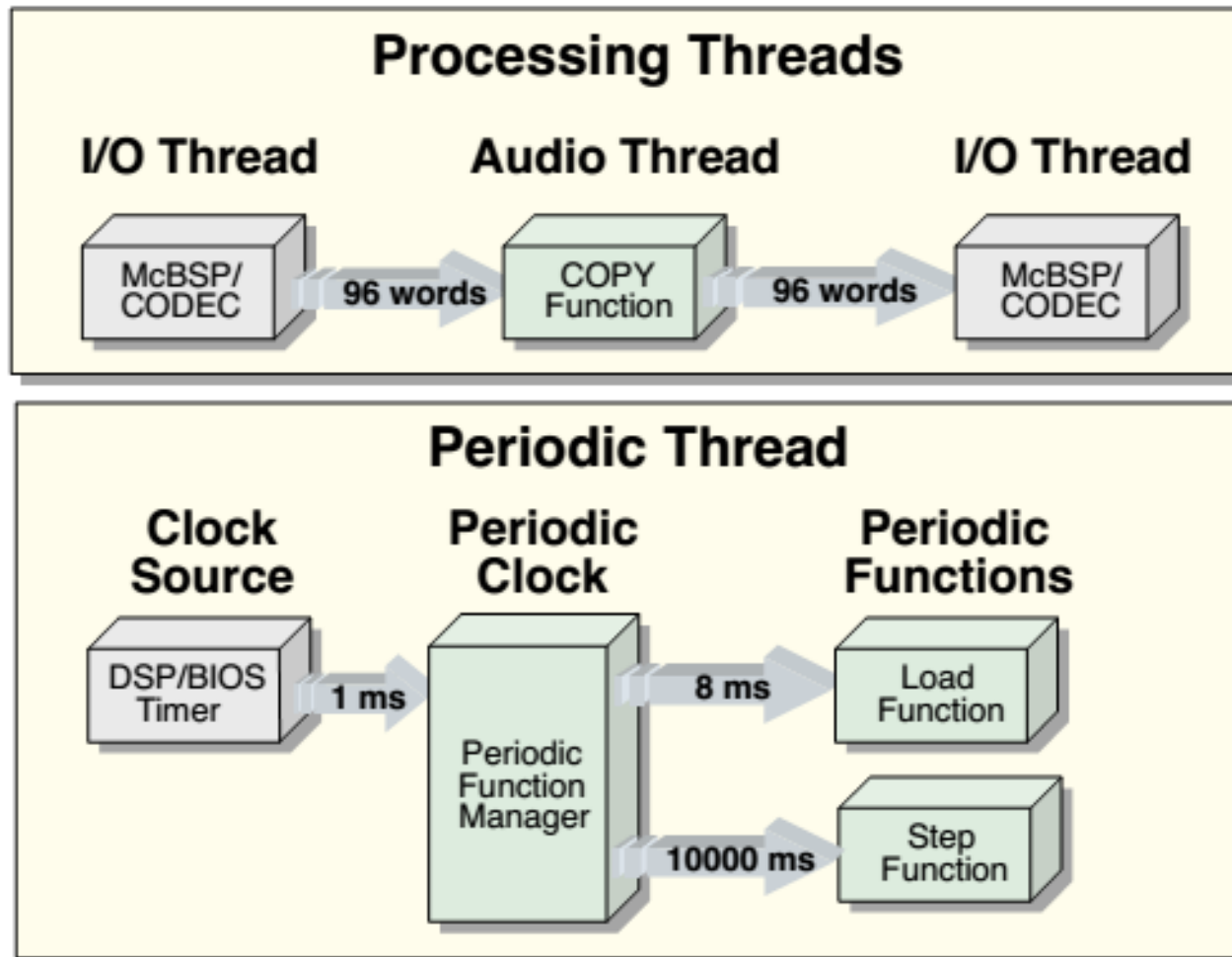
Estimated Data Size: 8744 Est. Min. Stack Size (MAUs): 528



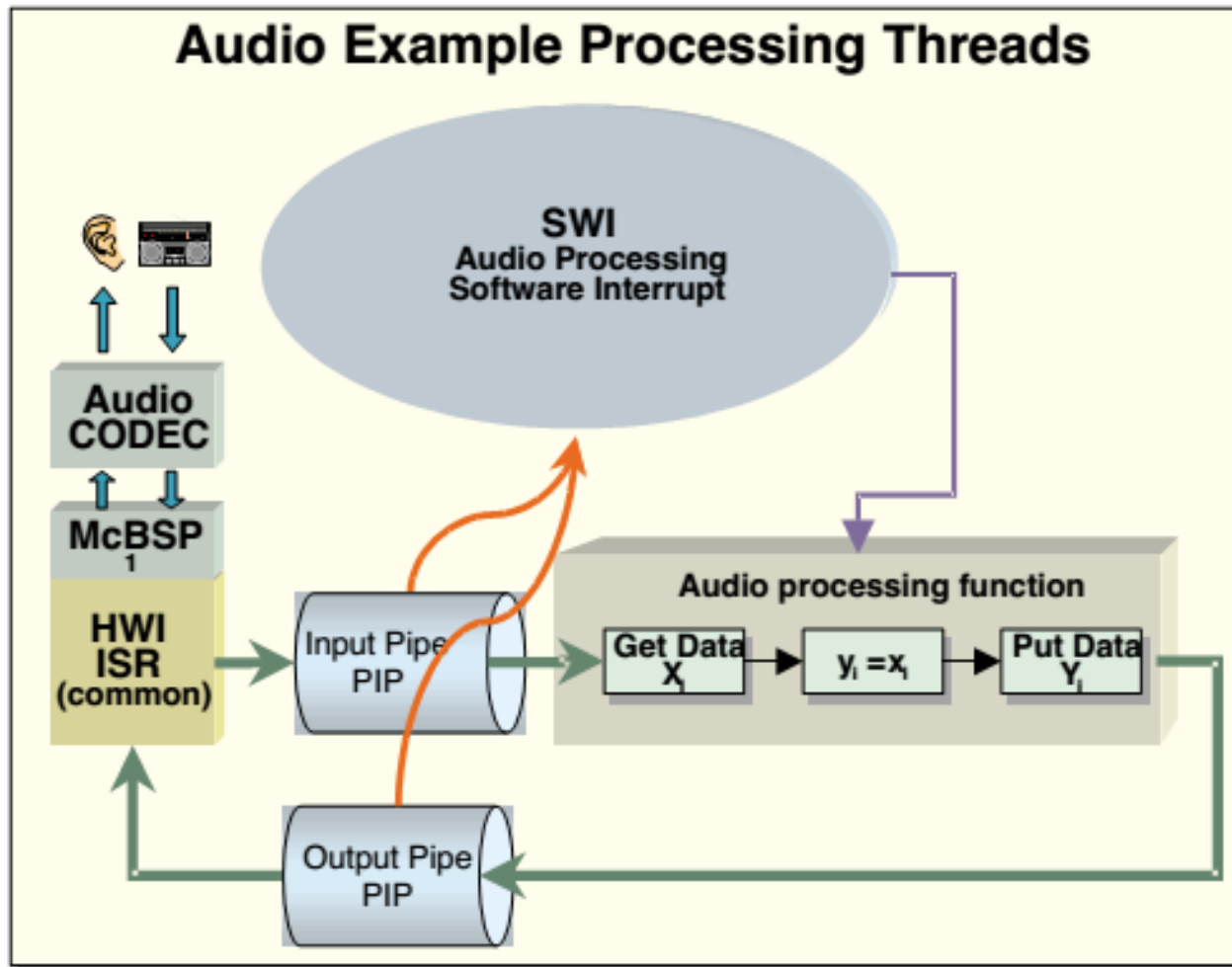
# Example: Audio Processing



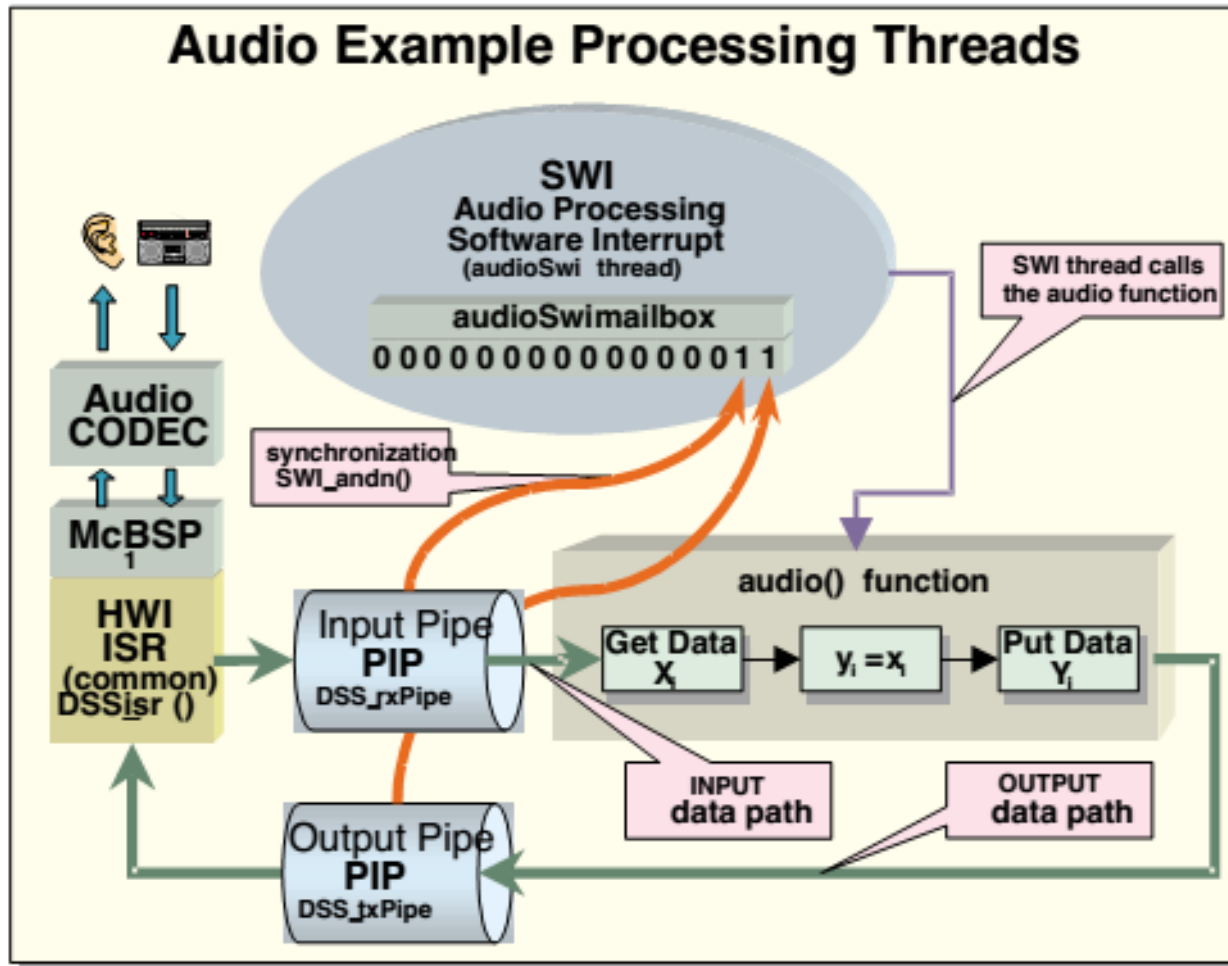
# Example: Audio Processing



# Example: Audio Processing



# Example: Audio Processing



# Frameworks

- SW-Architecture for DSP Systems
  - “Standard”-algorithms can be easily integrated
  - Based on system software
    - Chip support library
    - Board support library
- Framework is adaptable
  - Functionality
  - Number of channels
  - Dynamic/static configuration
  - Memory restrictions
  - . . .

# Reference Framework (TI)

Feature	Module		Object Creations		Language Support	
			STATIC	DYNAMIC	C routines	ASM routines
<b>Real Time Analysis and Data Capture</b>						
Event Logging	LOG	Message Log Manager	x		x	x
Statistics Accumulation	STS	Statistics Accumulator Manager	x		x	x
Trace Control	TRC	Trace Manager	x		x	x
File Streaming	HST	Host I/O Manager	x		x	x
Real Time Data Exchange	RTDX	Target to Host Communication Manager	x		x	x
<b>Hardware Abstraction</b>						
On-Chip Timer	CLK	System Clock Manager	x		x	x
Hardware Interrupts	HWI	Hardware Interrupt Manager	x			x
Static Memory Management	MEM *	Memory Segment Manager	x			
Dynamic Memory Management	MEM **	Memory Segment Manager		x	x	
<b>Device – Independent I/O</b>						
Data Pipes	PIP	Data Pipe Manager	x		x	x
Data Streams	SIO	Stream I/O Manager	x	x	x	

# Reference Framework (TI)

Feature	Module		Object Creations		Language Support	
			STATIC	DYNAMIC	C routines	ASM routines
<b>Execution Thread Management</b>						
Software Interrupts	SWI	Software Interrupt Manager	x	X	x	x
Periodic Functions	PRD	Periodic Function Manager	x		x	x
Tasks	TSK	Multitasking Manager	x	X	x	
Idle Loop	IDL	Idle Function and Processing Loop Manager	x		x	x
<b>Inter – Thread Communication and Synchronization</b>						
Semaphores	SEM	Semaphore Manager	x	X	x	
Resource Locks	LCK	Resource Lock Manager	x	X	x	
Mailboxes	MBX	Mailbox Manager	x	X	x	
Queues	QUE	Queue Manager	x	X	x	
<b>Other Services</b>						
Atomic Functions (optimized and non-preemptive)	ATM	Atomic Functions written in Assembly Language	N/A	N/A	x	
Error handling and program termination	SYS	System Services Manager	N/A	N/A	x	
* Using the <i>Configuration Tool</i> , memory segments are defined and named.						
** Once named, this module provides allocation and freeing services.						

# Conclusion



# Additional References

# Additional References

- Philip D. Lapsley:  
**DSP processor fundamentals: architectures and features**  
Berkeley Design Technology Inc, 1994. (Link: UB TUG)
- Lars Wanhammer: **DSP Integrated Circuits**  
Academic Press series in engineering, 1999, 1. ed.
- “How to Get Started With the DSP/BIOS Kernel” (SPRA782)
- “TMS320 DSP/BIOS User’s Guide” (SPRU423)
- “Code Composer Studio Development Tools” (SPRU509)
- “DSP/BIOS Technical Overview” (SPRA780, SPRA646)