

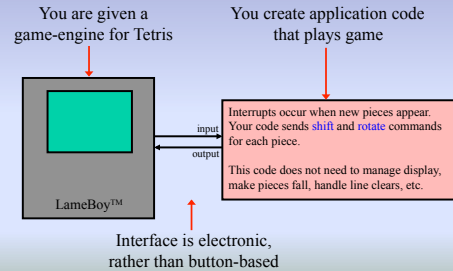
## Lab 8: Time-critical application code

- Inspiration for this lab: the ancient Game Boy
  - Real-time responsiveness was critical.
  - Had to balance computation and timely event handling.
  - Probably fun to design, develop, test.
- Our focus for this lab: **application software**
  - NOT the RTOS, graphical display, or game engine.



425 Lab8.1

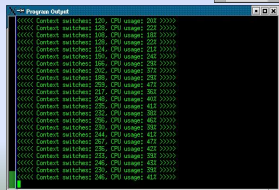
## Conceptual model



425 Lab8.2

## Simptris

- Simplified version of Tetris
  - Pieces consist of three blocks
  - Just two shapes: corners, straight pieces
  - Small board: 6 columns × 16 rows
- Score is simply total lines cleared
  - No bonus for clearing multiple lines
- Simptris is special mode in simulator
  - Type "simptris" to prompt
  - Opens additional window
  - Game displayed with ASCII characters



425 Lab8.3

## Operational details

- Inputs** to your system:
  - An interrupt occurs when each new piece appears
  - Global variables have details of location, piece type
- Outputs** from your system:
  - Movement commands: your code calls built-in rotate or shift functions
  - Constraint: next command cannot be sent until last command received
- Pieces fall faster until code can't keep up; high score wins
  - Limiting factor is fixed transmission delay of movement commands
  - For full credit: **clear at least 200 lines** at default tick frequency.



425 Lab8.4

## The interface: inputs

- The interrupts your system receives:
 

– reset	priority 0	} Emu86 interrupts
– tick	priority 1	
– keypress	priority 2	
– game over	priority 3	} Simptris interrupts
– new piece	priority 4	
– received command	priority 5	
– touchdown	priority 6	
– line clear	priority 7	
- Details are communicated via global variables
  - ID number, column, orientation of new piece
  - ID number of piece that touched down
  - Screen bitmaps of pieces that have touched down



425 Lab8.5

## The interface: outputs

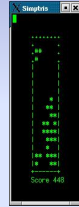
- The functions you can call (defined in simptris.s):
  - `void Slide_Piece(int id, int direction);` // 1=right, 0=left
  - `void Rotate_Piece(int id, int direction);` // 1=clockwise, 0=counterclockwise
  - `void Seed_Simptris(long seed);` // random number seed
  - `void Start_Simptris(void);`
- Dealing with transmission delay:
  - Wait for "command received" interrupt for previous command before calling `Slide_Piece()` or `Rotate_Piece()`, else behavior is undefined
  - Interrupt indicates "clear to send" rather than last command completed okay
    - Can't move piece into wall, for example
  - Recommendation: encapsulate communication details within task



425 Lab8.6

## The simulator

- Type "simpris" at Emu86> prompt and game display appears
  - Normal text output from your code will appear in the program output window as before
- You get reset, keypress, and timer ticks as before
  - Simpris interrupts are added in simpris mode
  - You decide which interrupts your code will pay attention to
    - Write required ISRs and handlers
    - Modify interrupt vector table
  - For each interrupt you want to ignore: write a minimal ISR.
    - Contents: save ax, send EOI command, restore ax, iret.
    - Impractical in simulator to modify IMR.



425 Lab8.7

## Lab requirements

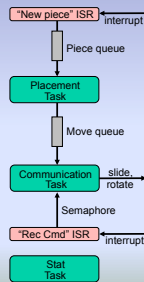
- Your application code must:
  - Use [your YAK kernel](#)
  - Accurately report CPU utilization and context switches every 20 ticks
  - Clear 200 lines at default tick frequency (with some seed)
  - Use just one random number seed per game
- Not an exercise in AI unless you choose to make it one
  - Fairly straightforward placement algorithms are adequate if the overhead of your RTOS code is low



425 Lab8.8

## Suggested organization

- Here's a starting point to consider:
  - Create **three tasks**:
    - One makes placement decisions, determines sequence of slide and rotate commands
    - One handles communication with Simpris
    - One handles statistics
  - Use **two queues**:
    - "Piece queue" buffers details about new pieces
    - "Move queue" buffers details about move commands
  - Use **one semaphore**:
    - Signal when next command can be sent
- Choose your own design, but use good design principles



425 Lab8.9

## Placement algorithms

- Very simple algorithm can clear 80+ lines (with right seed):
  - Straight pieces on one side, corners on the other
- Slightly more complicated algorithms work much better. **Example:**
  - Divide area into two halves, play each piece on selected half
  - If **both sides flat**, place pieces on nearest side unless imbalance too great
  - If **one side not flat**, place corner piece to make it flat
- Much more complicated algorithms are possible
  - Use features, interrupts in any way you wish
- Key measures to think about to achieve maximum scores:
  - Worst case number of moves for any piece
  - Worst case latency from new piece to first move command for piece



425 Lab8.10

## The friendly competition

- We will jointly pick a seed (next to last day of class)
- Everyone will run their code on that seed, [report results in class](#) on last day
- Special recognition (including Twinkies) awarded for
  - Highest score with that seed
  - Lowest score for code that cleared 200+ on another seed
  - Various noteworthy kernel "achievements" based on results reported in HW10
- For reference:
  - Minimum required: 200 lines
  - Maximum observed with current simulator: >450 lines
  - My code has been beaten



425 Lab8.11