EE310 Embedded Computer Systems

Lecture 10: Input and Output

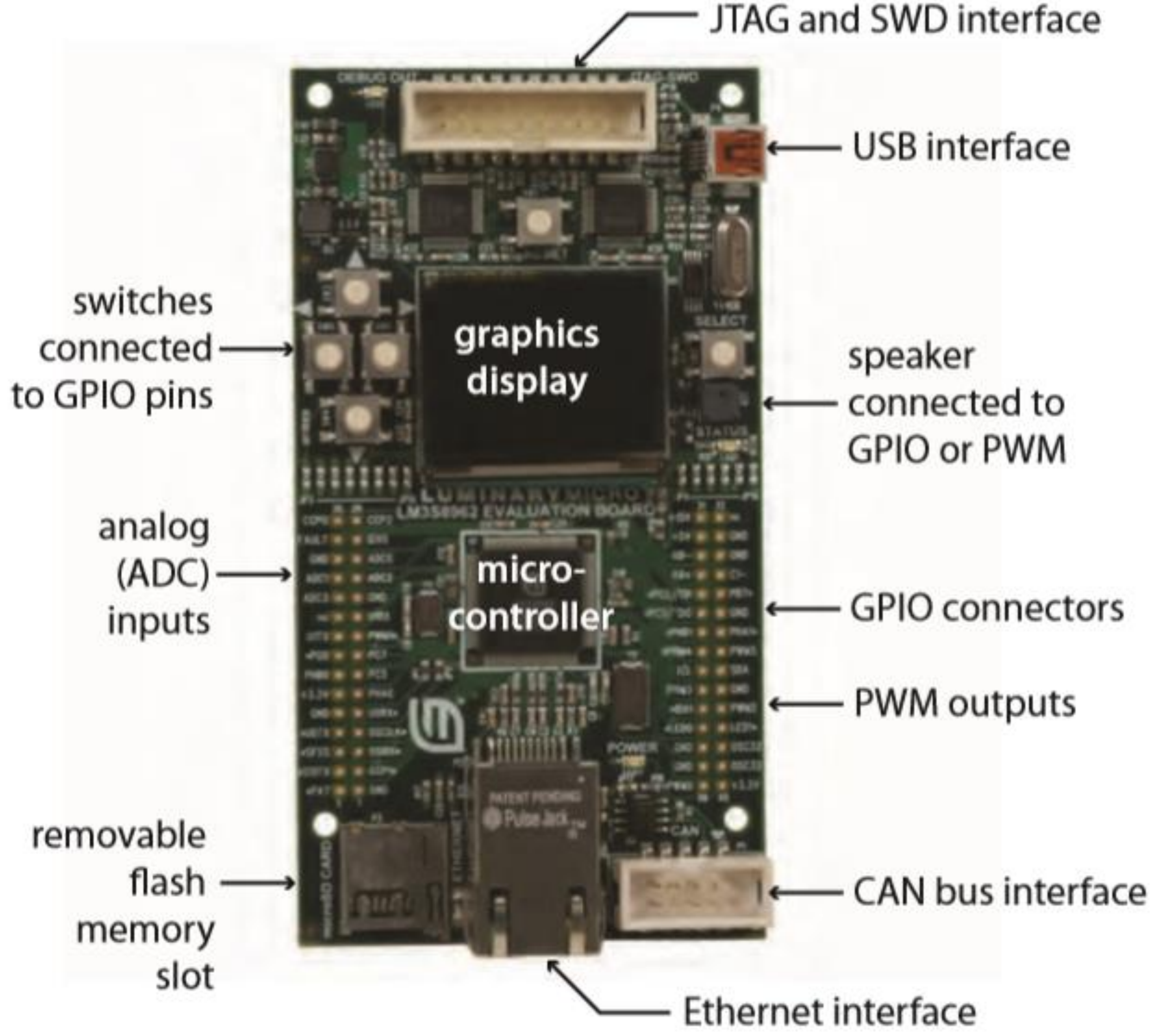
Dr. Suneel Kommuri
Suneel.Kommuri@xjtlu.edu.cn
Room EB324

Outline

- ✓ **I/O Hardware**
  - – Pulse width modulation
  - – General-purpose digital I/O
  - – Serial Interfaces
  - – Parallel Interfaces
  - – Buses
- ✓ **Software in a World**
  - – Interrupts and exceptions
  - – Atomicity
  - – Interrupt controllers
  - – Modeling Interrupts

1.1 I/O Hardware

- ✓ Embedded processors include I/O mechanisms on-chip
- ✓ Luminary Micro Stellaris
- ✓ Single-board computer



1.1.1 Pulse width modulation

- Delivers a variable amount of power efficiently to external hardware devices
  - example: to control the speed of electric motors
- PWM hardware uses only digital circuits and easy to integrate on chip with a microcontroller
- If duty cycle is 100%, then the voltage is always high
- Many microcontrollers provide PWM peripheral devices
  - programmer typically writes a value to *memory-mapped register*

1.1.2 General-Purpose Digital I/O (GPIO)

- Many embedded processors have number of general purpose I/O pins
  - enables software to either read or write voltage levels (logical 0 or 1)
  - if V_{DD} in *active high logic*, voltage close to V_{DD} represents logical one, voltage near zero represents a logical zero
  - in *active low logic*, these interpretations are reversed
- In many designs, a GPIO pin may be configured to be an output
- If interfacing hardware to GPIO pins, designer needs to understand specifications of the device – particularly, voltage and current levels
- If GPIO produces V_{DD} and device with resistance R ohms, then output current will be $I = V_{DD}/R$

1.1.2 General-Purpose Digital I/O (GPIO)

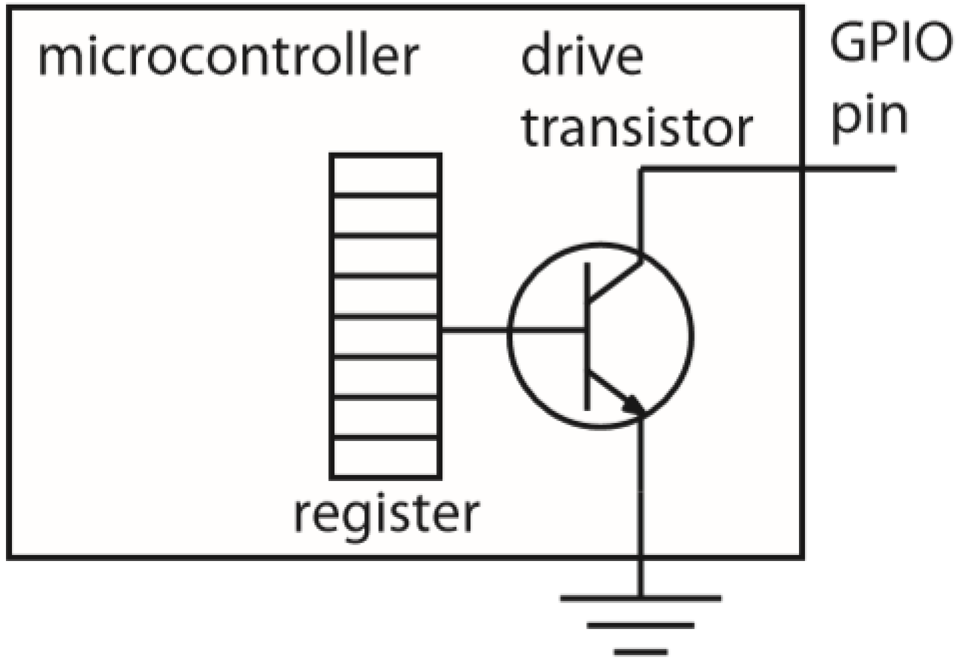
- Maintain *electrical isolation* between processor and external devices
  - external devices may have noisy electrical characteristics
  - useful strategy is to divide circuit into *electrical domains*

- GPIO outputs may use *open collector* circuits
  - writing logical one into the register turns on the transistor
  - writing logical zero into the register turns off the transistor

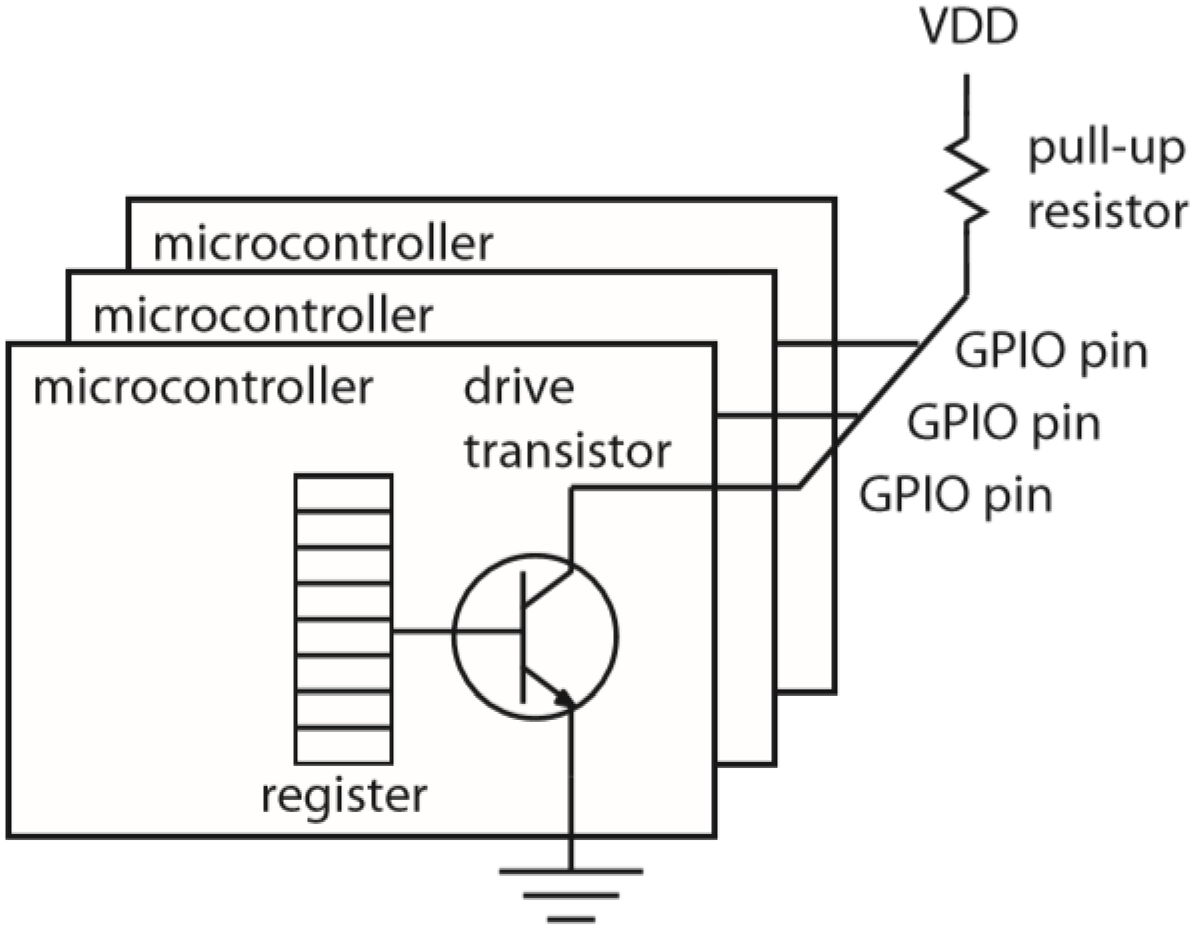
- A number of open collector interfaces may be connected
  - shared lime is connected to a *pull-up resistor*, brings voltage to V_{DD}

- GPIO outputs also be realized with tristate logic – simply turned off

1.1.2 General-Purpose Digital I/O



An open collector circuit for a GPIO pin



A number of open collector circuits wired together

1.1.3 Serial Interfaces

- Key constraints – small packages and low power consumption

- The number of pins on the processor integrated circuit is limited
  - each pin must be used efficiently, wires must also be used efficiently
  - one way to use efficiently is to send information over them serially as sequences of bits, such interface is called *serial interface*

- RS-232 – sender and receiver must agree on a transmission rate
  - sender initiates transmission of a byte with a start bit
  - sender then clocks out the sequence of bits at the agreed-upon rate
  - receivers clock resets upon receiving the start bit

1.1.3 Serial Interfaces

- RS-232 connection may be provided via DB-9 connector
- USB is electrically simpler than RS-232, uses robust connectors
- JTAG (Joint Test Action Group) serial interface is widely implemented in embedded processors

DB-9 serial port

DB-25 parallel port

USB

IEEE 488

1.1.4 Parallel Interfaces

- A *serial interface* sends or receives sequence of bits sequentially over a single line
- *Parallel interface* uses multiple lines to simultaneously send bits
  - each line is also serial interface, but logical grouping makes it parallel
- With careful programming, a group of GPIO pins can be used together to realize a parallel interface
- Parallel interfaces deliver higher performance than serial interfaces
  - because more wires are used for the interconnection

1.1.5 Buses

- A *bus* is an interface shared among multiple devices
- Buses can be serial (USB) or parallel interfaces
- Any bus architecture must include media-access control (MAC) to arbitrate competing accesses
  - MAC protocol has single bus master that interrogates bus slaves
  - USB uses such a mechanism
- Alternative is time-triggered bus, devices are assigned time slots during which they can transmit
- *Token ring*, devices must acquire token before they use shared medium

1.2 Sequential Software in a Concurrent World

- If software interacts with external world, execution time may be effected
- Software is intrinsically sequential, executes as fast as possible
- The physical world is concurrent, many things happen at once and will be determined by their physical properties
  – bridging this mismatch is one of the major challenges

1.2.1 Interrupts and Exceptions

- An *interrupt* is a mechanism for pausing execution of current code
  - executing pre-defined code sequence: *interrupt service routine* (ISR)

- Three kinds of events may trigger an interrupt
  - *hardware interrupt*, hardware changes voltage level on interrupt line
  - *software interrupt*, program triggers the interrupt by an instruction
  - *exception*, interrupt is triggered by internal hardware that detects fault

- Hardware decides whether to respond on occurrence of interrupt trigger
  - if interrupts are disabled, it will not respond
  - it varies by processor for enabling or disabling interrupts

Timers

- Microcontrollers always include some peripheral devices, *timers*
- A *programmable interval timer (PIT),* simply counts down to zero
- Initial value is set and when it hits zero, PIT raises an interrupt request
- A timer might be set up to trigger repeatedly without to be reset
  - such repeated triggers will be more precisely periodic
- If timer reaches zero at a time when interrupts happen to be disabled
  - there will be a delay before ISR gets invoked

Timers

```c
volatile uint timerCount = 0;
void countDown(void) {
    if (timerCount != 0) {
        timerCount--;
    }
}
```

```c
SysTickPeriodSet(SysCtlClockGet() / 1000);
SysTickIntRegister(&countDown);
SysTickEnable();
SysTickIntEnable();
```

```c
int main(void) {
    timerCount = 2000;
    ... initialization code from above ...
    while(timerCount != 0) {
        ... code to run for 2 seconds ...
    }
}
```

1.2.2 Atomicity

- An ISR can be invoked between any two instructions of main program
  – term *atomic* comes from Greek work for “indivisible”

- It may be safe to assume each assembly instruction is atomic

```
timerCount = 2000;
```

This statement may take more than one instruction cycle to execute

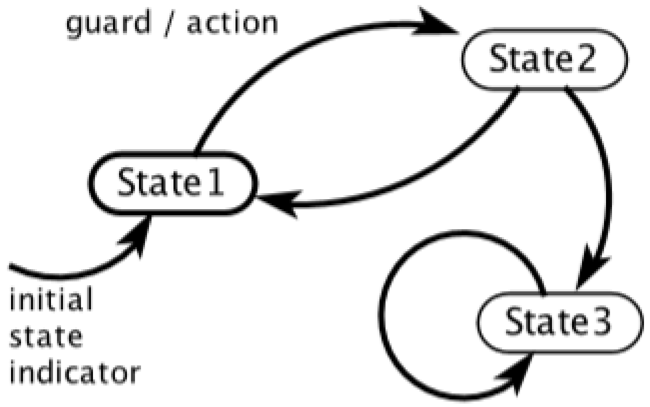
1.2.3 Interrupt Controllers

- *Interrupt controller* is the logic in processor that handles interrupts
  - supports number of interrupts and priority levels
- Each interrupt has an interrupt vector (address of an ISR or index into an array – *interrupt vector table* contains addressee of all ISRs)
- When an interrupt is asserted by changing the voltage on a pin
  - the response may be either *level triggered* or *edge triggered*
- *Level triggered,* hold voltage on the line until gets acknowledgment
- *Edge triggered*, changes the voltage for only a short-time

1.2.4 Modelling Interrupts

- The behavior of interrupts can be quite difficult to understand and many catastrophic failures are caused by unexpected behaviors

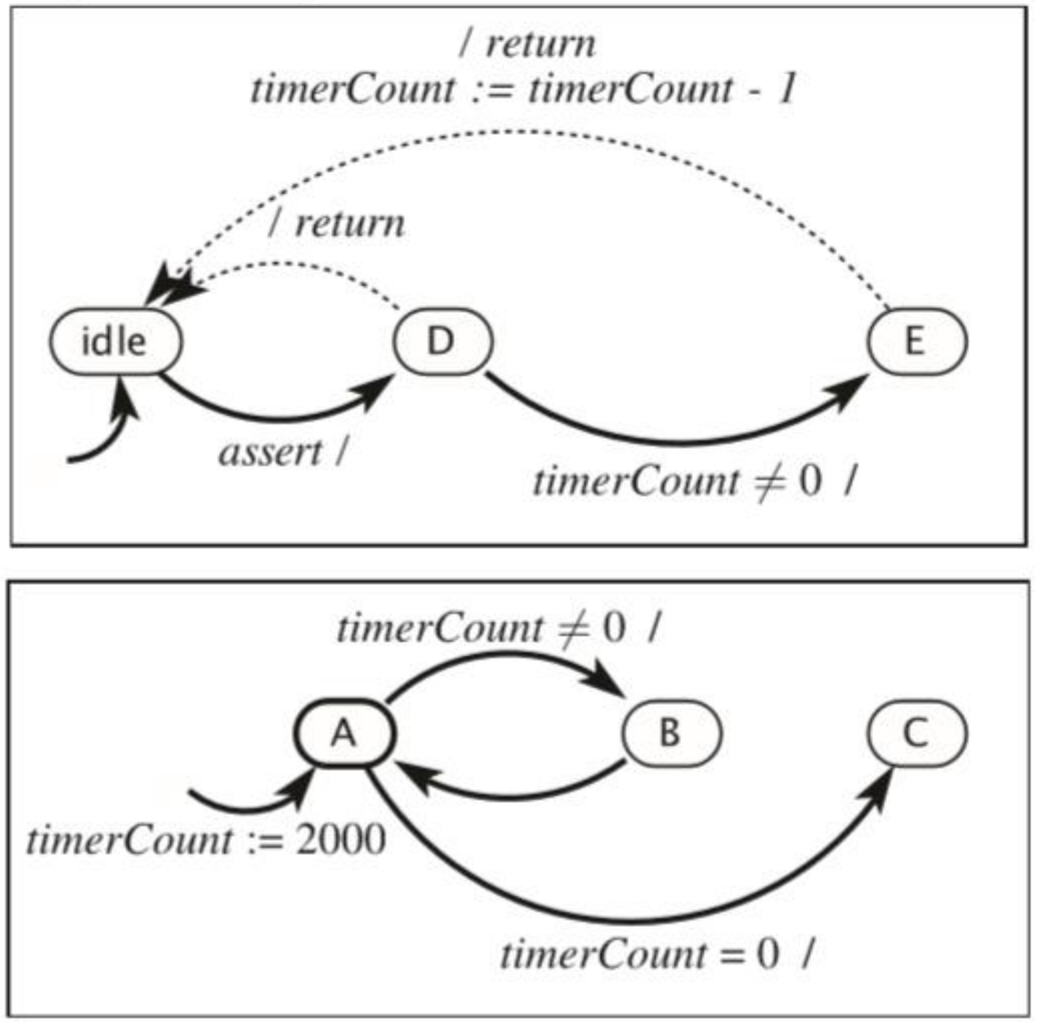
- The logic of interrupt controllers describes in processors imprecisely, leaving many possible behaviors unspecified
  - possible way to make this logic precise is to model as *FSM*
  - *finite state machines,* the set states of possible states is finite

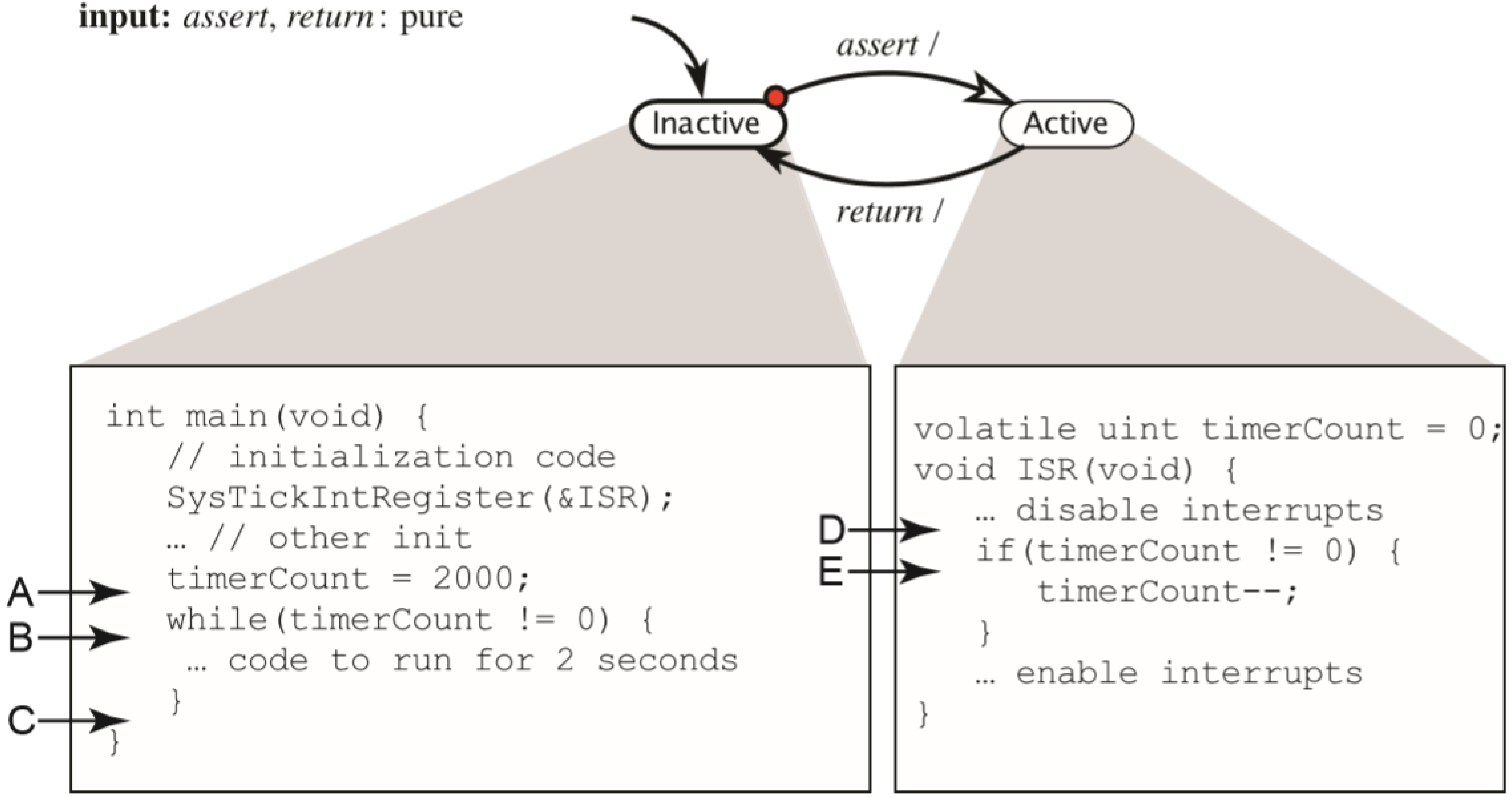


$$States = \{\text{State1}, \text{State2}, \text{State3}\}.$$

```
    volatile uint timerCount = 0;
    void ISR(void) {
       … disable interrupts
D→     if(timerCount != 0) {
E→        timerCount--;
       }
       … enable interrupts
    }
    int main(void) {
       // initialization code
       SysTickIntRegister(&ISR);
       … // other init
       timerCount = 2000;
A→     while(timerCount != 0) {
B→      … code to run for 2 seconds
       }
    }
C→  … whatever comes next
```

variables: *timerCount*: `uint`
input: *assert* : pure
output: *return* : pure

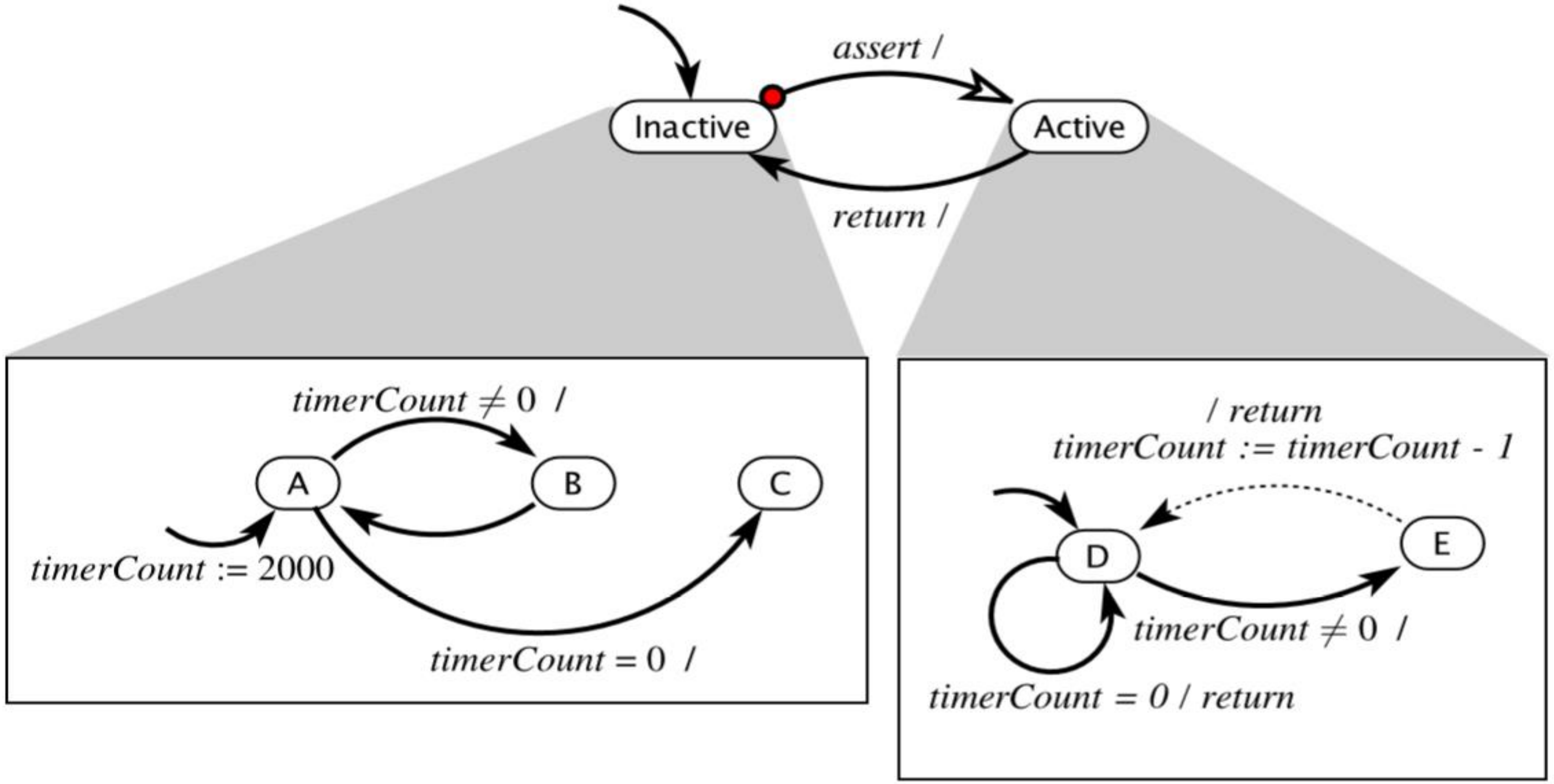


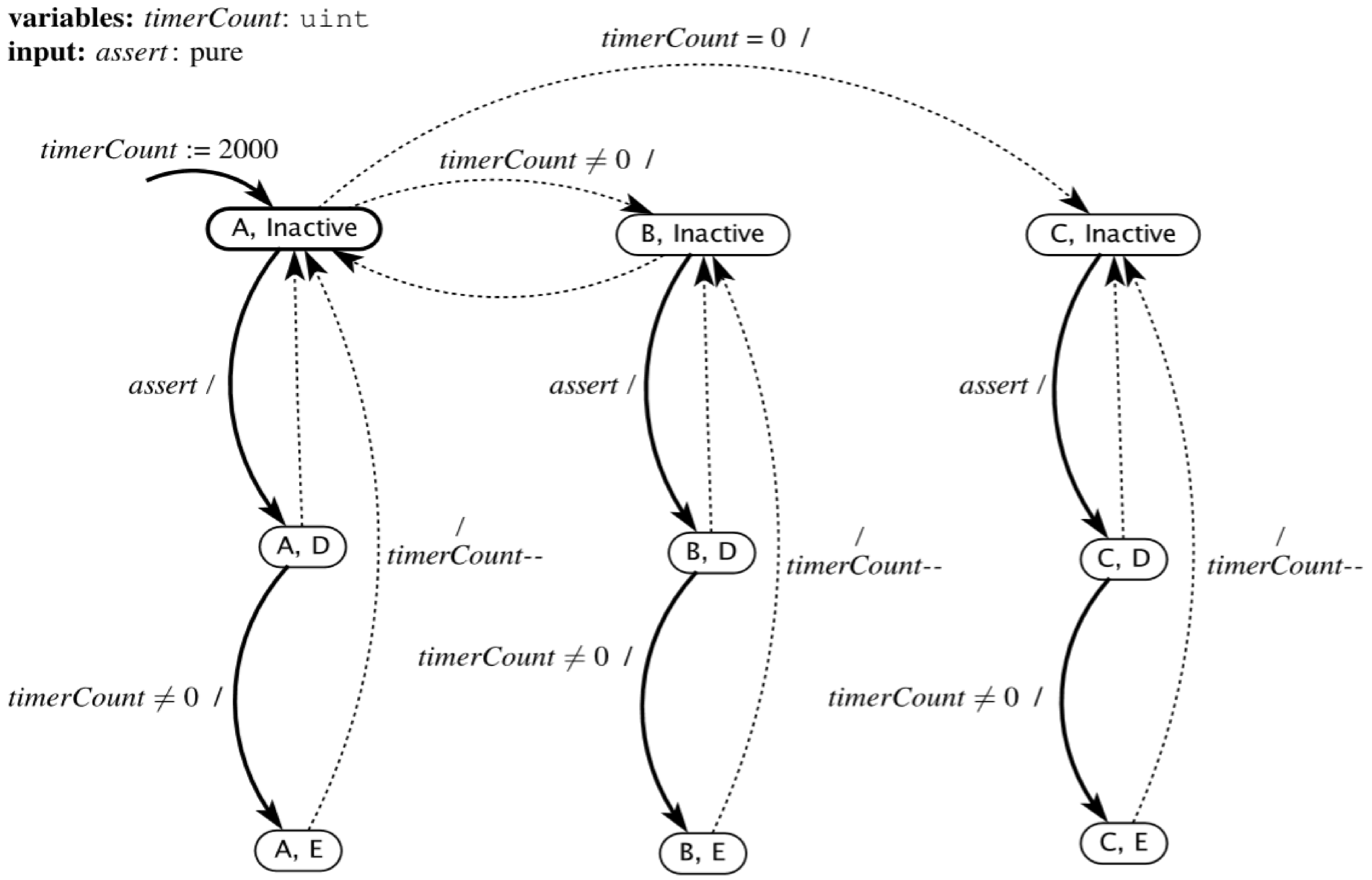


```
int main(void) {
   // initialization code
   SysTickIntRegister(&ISR);
   … // other init
   timerCount = 2000;
   while(timerCount != 0) {
    … code to run for 2 seconds
   }
}
```

```
volatile uint timerCount = 0;
void ISR(void) {
   … disable interrupts
   if(timerCount != 0) {
      timerCount--;
   }
   … enable interrupts
}
```

variables: *timerCount*: `uint`
input: *assert*: pure, *return*: pure
output: *return*: pure




variables: timerCount: uint
input: assert: pure
timerCount = 0 /
timerCount := 2000
timerCount ≠ 0 /
A, Inactive
B, Inactive
C, Inactive
assert /
assert /
assert /
A, D
B, D
C, D
/ timerCount--
/ timerCount--
/ timerCount--
timerCount ≠ 0 /
timerCount ≠ 0 /
timerCount ≠ 0 /
A, E
B, E
C, E

Summary

- ***PWM*** signal rapidly switches between high and low at some fixed frequency
- Embedded processors have number of GPIO, enable the software either to read or write voltage levels
- ***JTAG*** serial interface is widely used in embedded processors
- ***Parallel interface*** uses multiple lines to simultaneously send bits
- ***Bus*** is an interface shared among multiple devices
- ***Interrupt*** is used to pause execution of program code

See you in the next class (May 21st)

The End