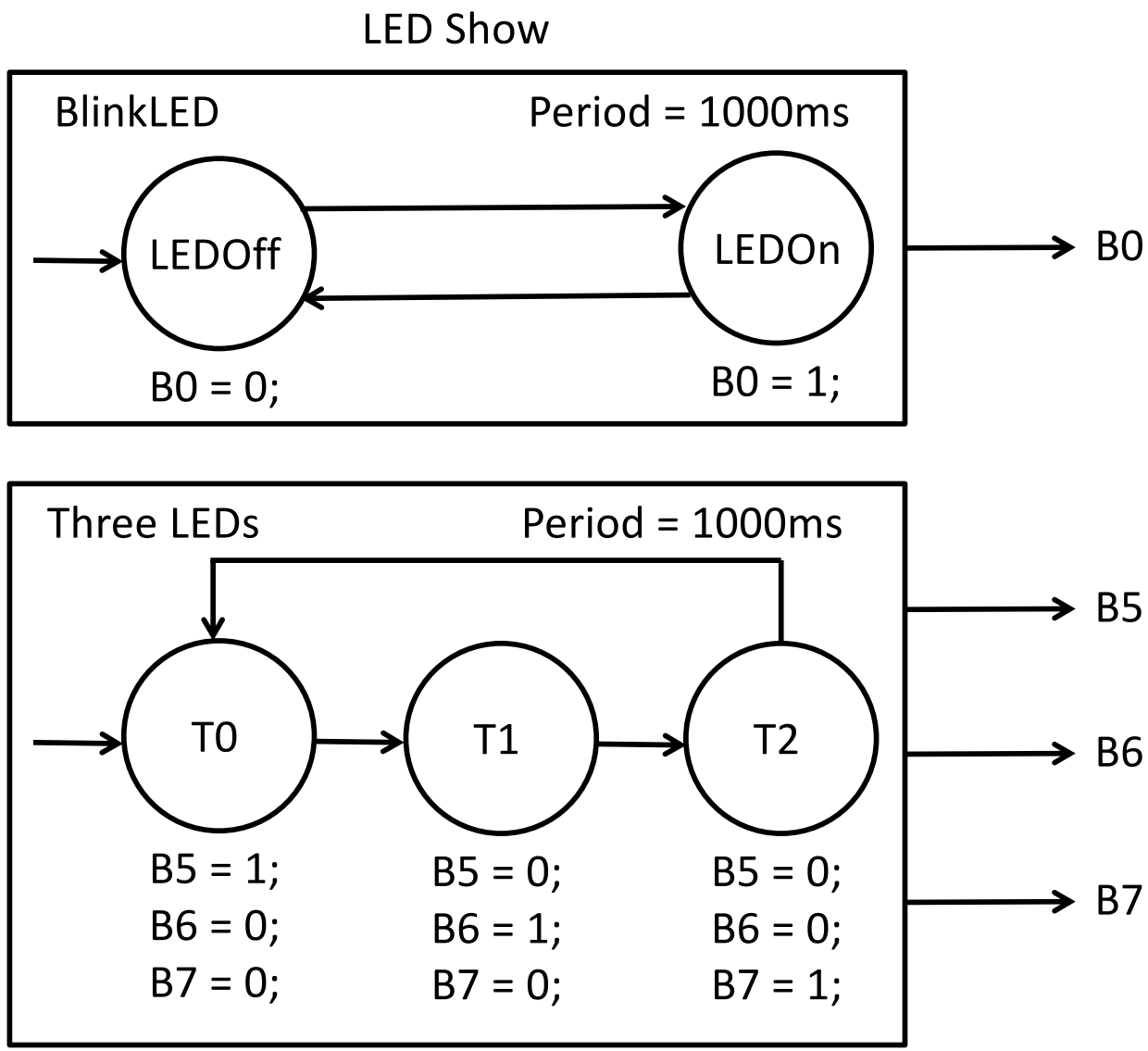
Concurrent Synchronous State Machines and Conversion to C

Overview

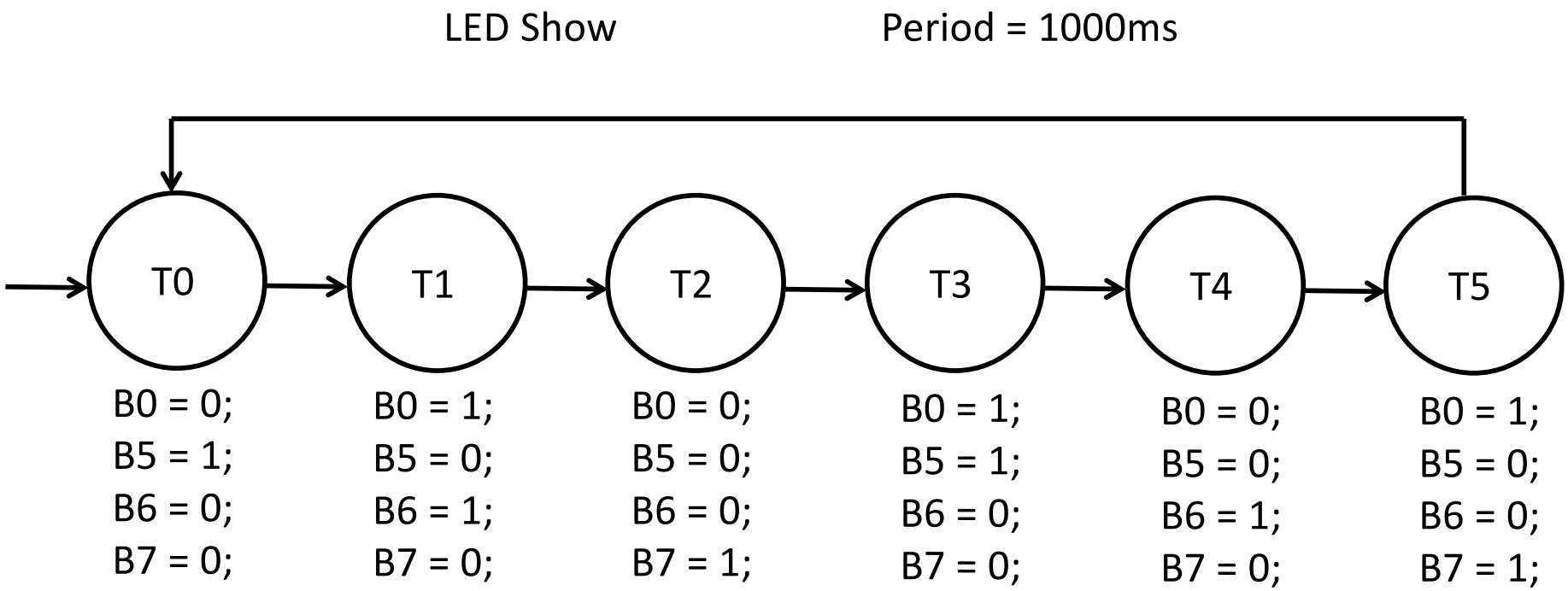
- Assumption For Today:
  - No communication between tasks

1. All tasks have the same period
2. Tasks have different periods
3. Task Struct and Cooperative Scheduler

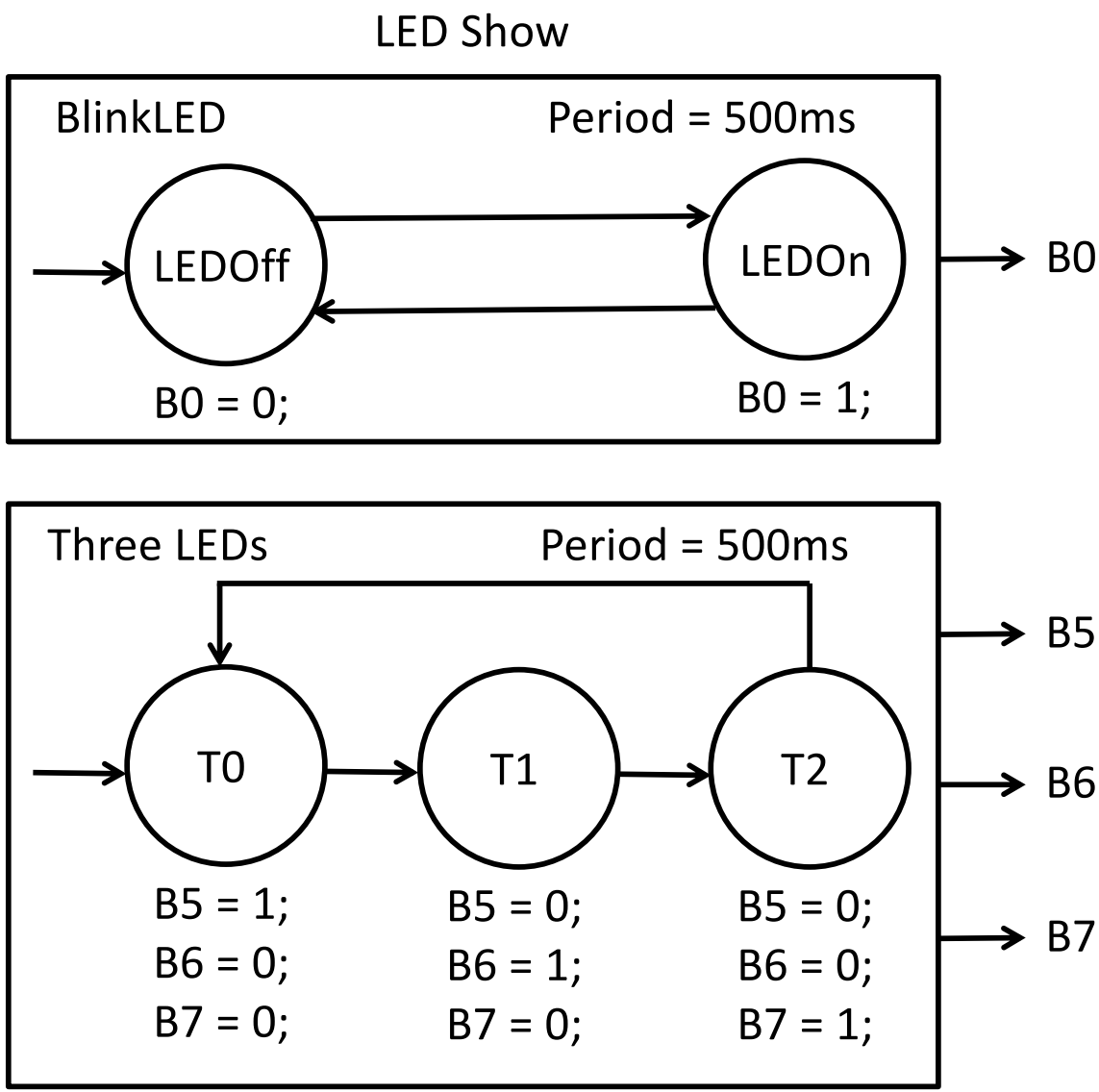
Independent Operations



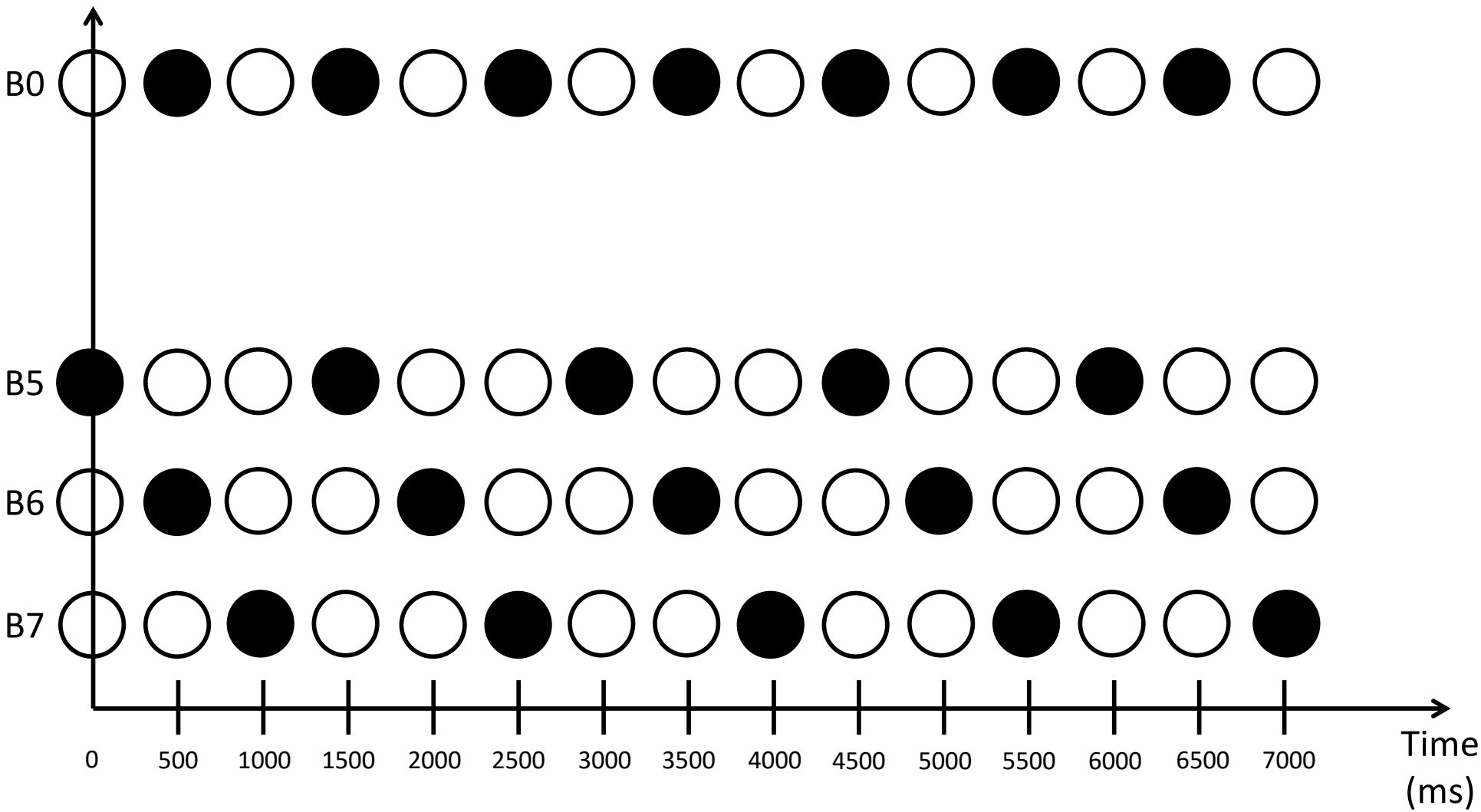
Single SynchSM Requires 6 States



Example: Same Period



Output State After Each Tick



C Code

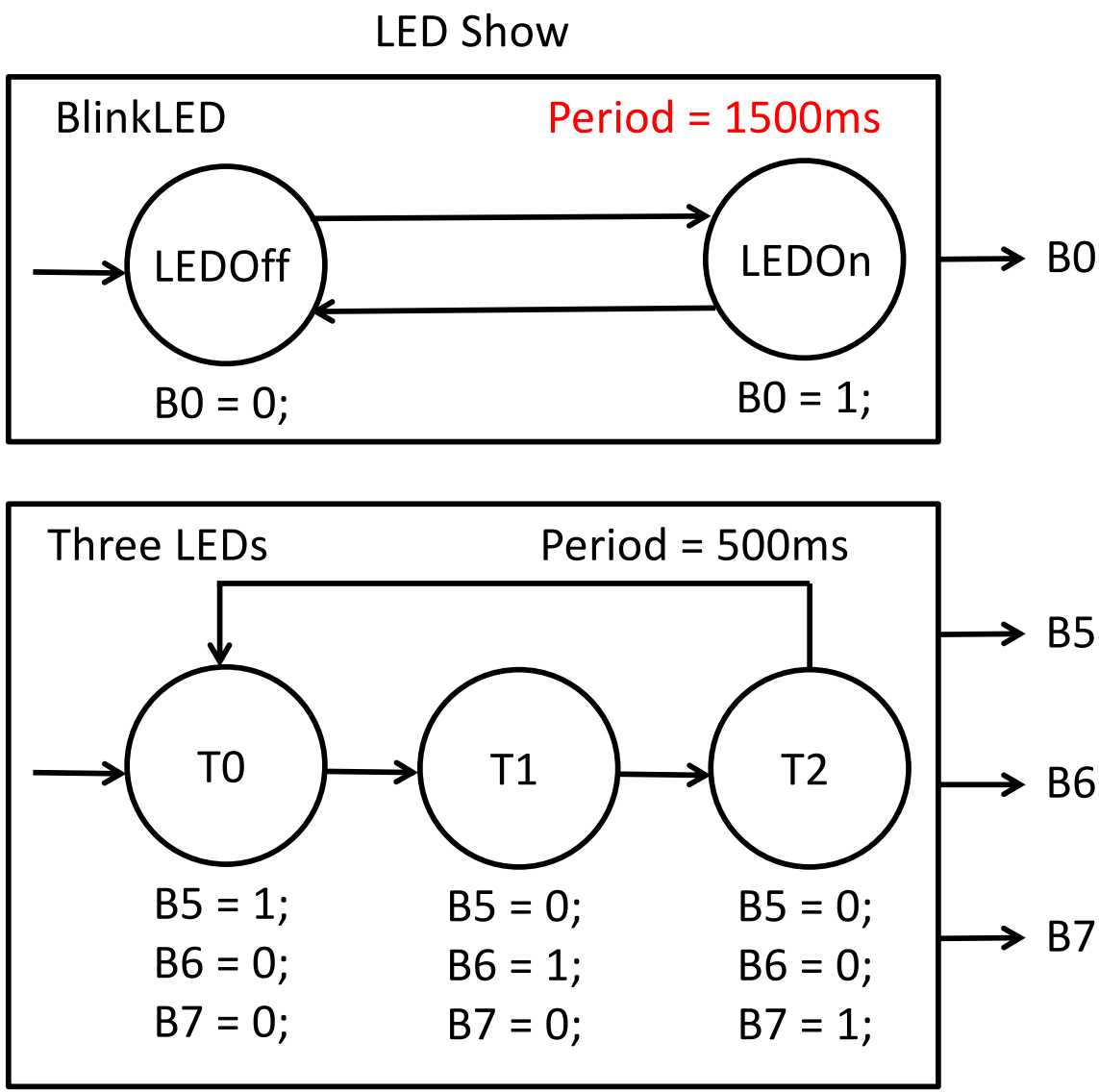
```
void main() {
    B = 0;                          // Init outputs
    TimerSet(500);
    TimerOn();
    BL_State = BL_SMStart;
    TL_State = TL_SMStart;
    while (1) {
        TickFct_BlinkLed();         // Tick the BlinkLed synchSM
        TickFct_ThreeLeds();        // Tick the ThreeLeds synchSM
        while (!TimerFlag) {}       // Wait for timer period
        TimerFlag = 0;              // Lower flag raised by timer
    }
}
```

Overview

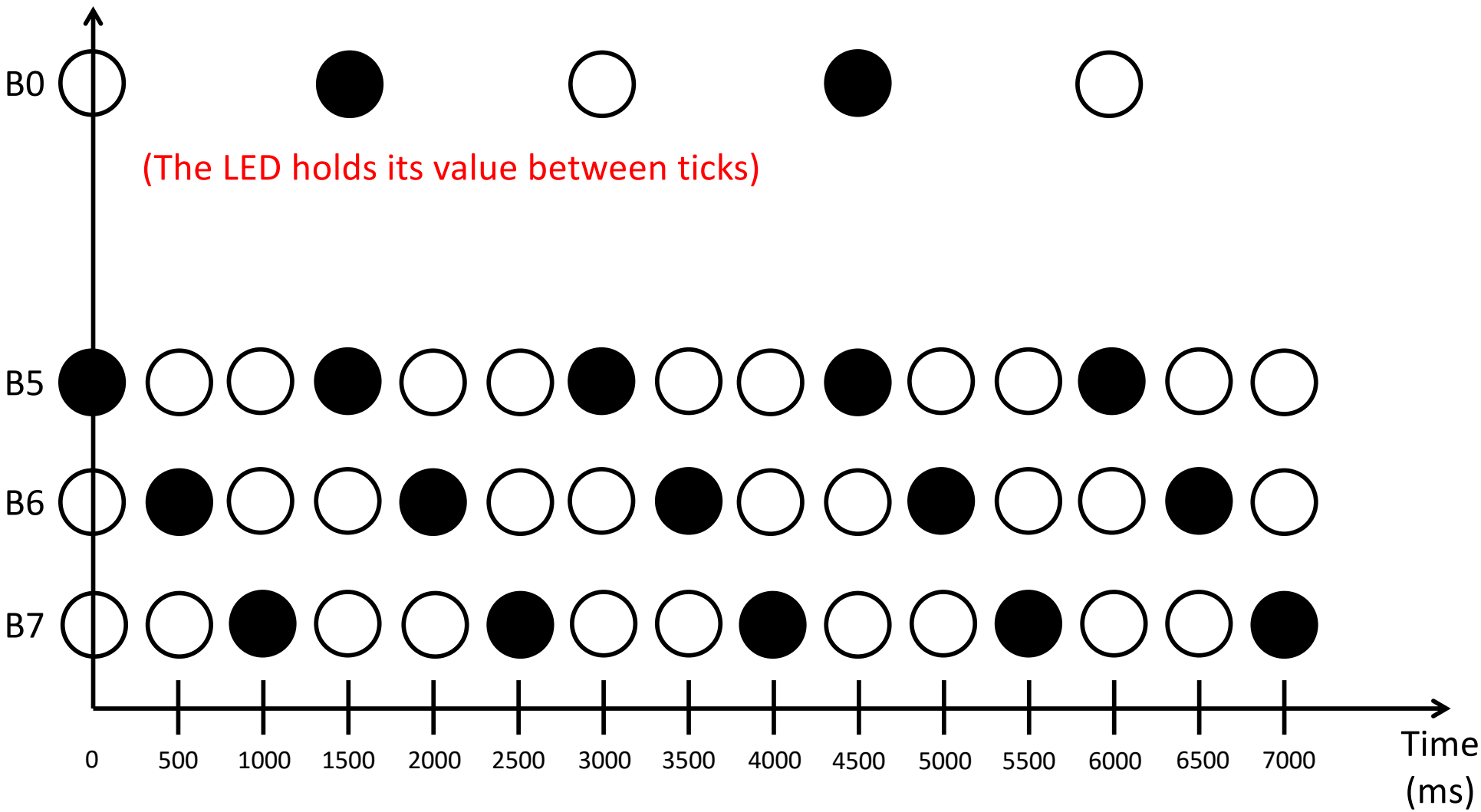
- Assumption For Today:
  - No communication between tasks

1. All tasks have the same period
2. Tasks have different periods
3. Task Struct and Cooperative Scheduler

Example: Different Periods



Output State After Each Tick



C Code

```
void main() {
    unsigned long BL_elapsedTime = 0;
    unsigned long TL_elapsedTime = 0;
    const unsigned long timerPeriod = 100;

    B = 0;
    TimerSet(100);
    TimerOn();
    BL_State = BL_SMStart;
    TL_State = TL_SMStart;
    while (1) {
        if (BL_elapsedTime >= 1500) {          // 1500 ms period
            TickFct_BlinkLed();                // Execute one tick of the BlinkLed synchSM
            BL_elapsedTime = 0;
        }
        if (TL_elapsedTime >= 500) {           // 500 ms period
            TickFct_ThreeLeds();               // Execute one tick of the ThreeLeds synchSM
            TL_elapsedTime = 0;
        }
        while (!TimerFlag) {}                  // Wait for timer period
        TimerFlag = 0;                         // Lower flag raised by timer
        BL_elapsedTime += timerPeriod;
        TL_elapsedTime += timerPeriod;
    }
}
```

timerPeriod must evenly divide all task periods
(you can always use the GCD)

Coding Style

```
void main() {
    unsigned long BL_elapsedTime = 0;
    unsigned long TL_elapsedTime = 0;
    const unsigned long timerPeriod = 100;

    B = 0;
    TimerSet(100);
    TimerOn();
    BL_State = BL_SMStart;
    TL_State = TL_SMStart;
    while (1) {
        if (BL_elapsedTime >= 1500) {
            TickFct_BlinkLed();
            BL_elapsedTime = 0;
        }
        if (TL_elapsedTime >= 500) {
            TickFct_ThreeLeds();
            TL_elapsedTime = 0;
        }
        while (!TimerFlag) {}
        TimerFlag = 0;
        BL_elapsedTime += timerPeriod;
        TL_elapsedTime += timerPeriod;
    }
}
```

One elapsed Time variable per task

One if-statement per task

One increment per task

Coding Style

```
void main() {
    unsigned long BL_elapsedTime = 0;
    unsigned long TL_elapsedTime = 0;
    const unsigned long timerPeriod = 100;

    B = 0;
    TimerSet(100);
    TimerOn();
    BL_State = BL_SMStart;
    TL_State = TL_SMStart;
    while (1) {
        if (BL_elapsedTime >= 1500) {
            TickFct_BlinkLed();
            BL_elapsedTime = 0;
        }
        if (TL_elapsedTime >= 500) {
            TickFct_ThreeLeds();
            TL_elapsedTime = 0;
        }
        while (!TimerFlag) {}
        TimerFlag = 0;
        BL_elapsedTime += timerPeriod;
        TL_elapsedTime += timerPeriod;
    }
}
```

The only differences are:

- Initial states
- Task periods
- Tick() functions

Overview

- Assumption For Today:
  - No communication between tasks

1. All tasks have the same period
2. Tasks have different periods
3. Task Struct and Cooperative Scheduler

Task Struct

```
typedef struct task {
    int state;                      // Task's current state
    unsigned long period;           // Task period
    unsigned long elapsedTime;      // Time elapsed since last task tick
    int (*TickFct)(int);            // Task tick function
} task;
```

Task Struct

```
typedef struct task {
    int state;                      // Task's current state
    unsigned long period;           // Task period
    unsigned long elapsedTime;      // Time elapsed since last task tick
    int (*TickFct)(int);            // Task tick function
} task;
```

Yes, that really is a function pointer
(and I'm not crazy!)

C Code with Task Struct (1/3)

```
typedef struct task {
        int state;                          // Task's current state
        unsigned long period;               // Task period
        unsigned long elapsedTime;          // Time elapsed since last task tick
        int (*TickFct)(int);                // Task tick function
} task;

task tasks[V];
const unsigned short tasksNum = V;

enum BL_States { BL_SMStart, BL_S1 };
int TickFct_BlinkLed(int state) { … return state; }

enum TL_States { TL_SMStart, TL_S1, TL_S2, TL_S3 };
int TickFct_ThreeLeds(int state) { … return state; }

…
```

To add an extra task to the system, increment V in your code

The Tick() functions must now return the updated state

C Code with Task Struct (2/3)

```
int main() {
    unsigned char i = 0;
    tasks[i].state = BL_SMStart;
    tasks[i].period = 500;
    tasks[i].elapsedTime = 0;
    tasks[i].TickFct = &TickFct_BlinkLed;
    i++;
    tasks[i].state = TL_SMStart;
    tasks[i].period = 1500;
    tasks[i].elapsedTime = 0;
    tasks[i].TickFct = &TickFct_ThreeLeds;

    TimerSet(100);
    TimerOn();

    while(1) { }
    return 0;
}
```

Initialize the Tick() function pointers

You need to initialize each new task in your system

What happened to the Ticks?

C Code with Task Struct (3/3)

```
task tasks[V];
const unsigned short tasksNum = V;

void TimerISR() {
	unsigned char I;
	for (i = 0; i < tasksNum; ++i) {
		if ( tasks[i].elapsedTime >= tasks[i].period ) {
			tasks[i].state = tasks[i].TickFct(tasks[i].state);
			tasks[i].elapsedTime = 0;
		}
		tasks[i].elapsedTime += tasksPeriod;
	}
}
```

Invoke the task's Tick() function via the function pointer

This code NEVER changes if you add or remove a task.

- The only thing you do is change the value of tasksNum, which is a constant variable declared in the global scope.

Minor Enhancements

```
int main() {
    unsigned char i = 0;
    tasks[i].state = BL_SMStart;
    tasks[i].period = 500;
    tasks[i].elapsedTime = 0;
    tasks[i].TickFct = &TickFct_BlinkLed;
    i++;
    tasks[i].state = TL_SMStart;
    tasks[i].period = 1500;
    tasks[i].elapsedTime = 0;
    tasks[i].TickFct = &TickFct_ThreeLeds;

    TimerSet(100);
    TimerOn();

    while(1) { Sleep(); }
    return 0;
}
```

Could be replaced with globally declared constants

Go into a low power mode between timer interrupts