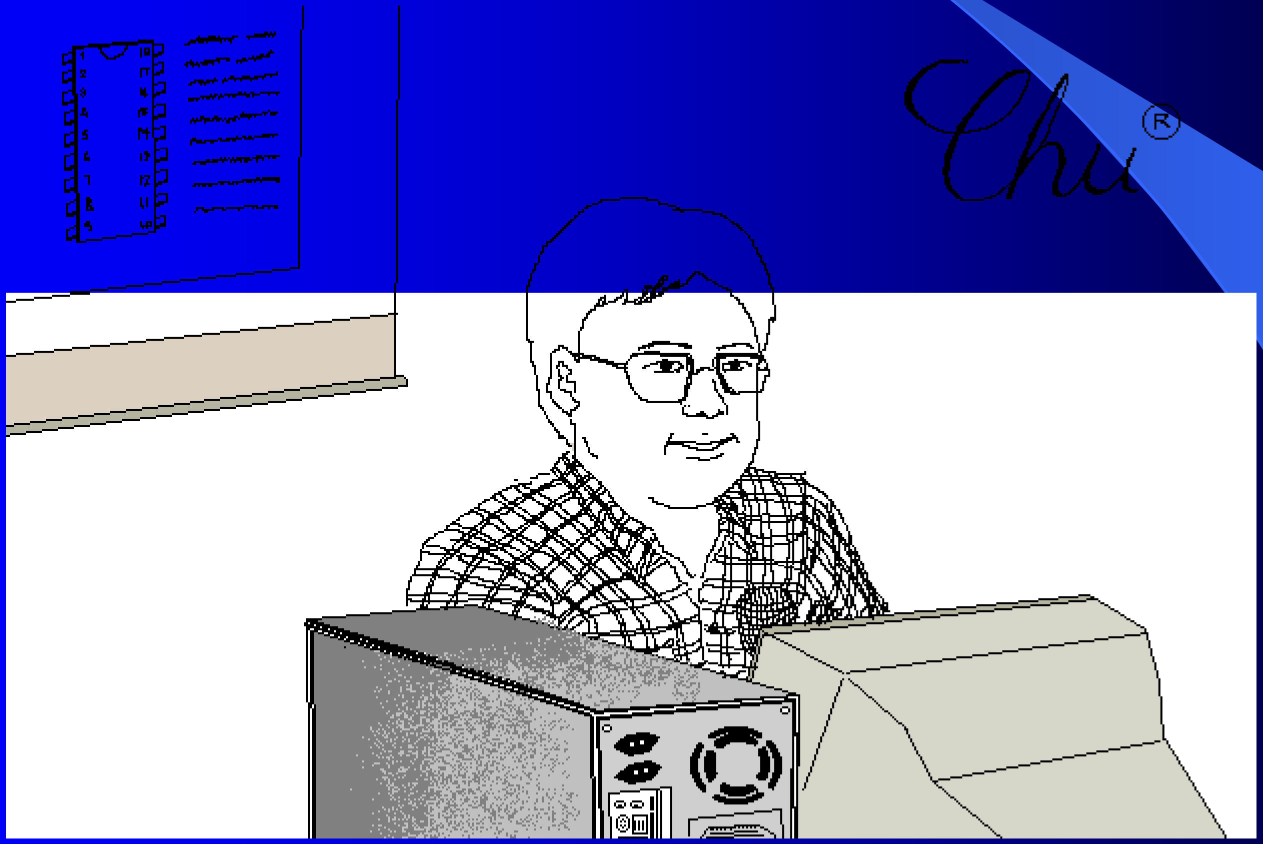
Background/Foreground Event-driven Finite State Machine Programming

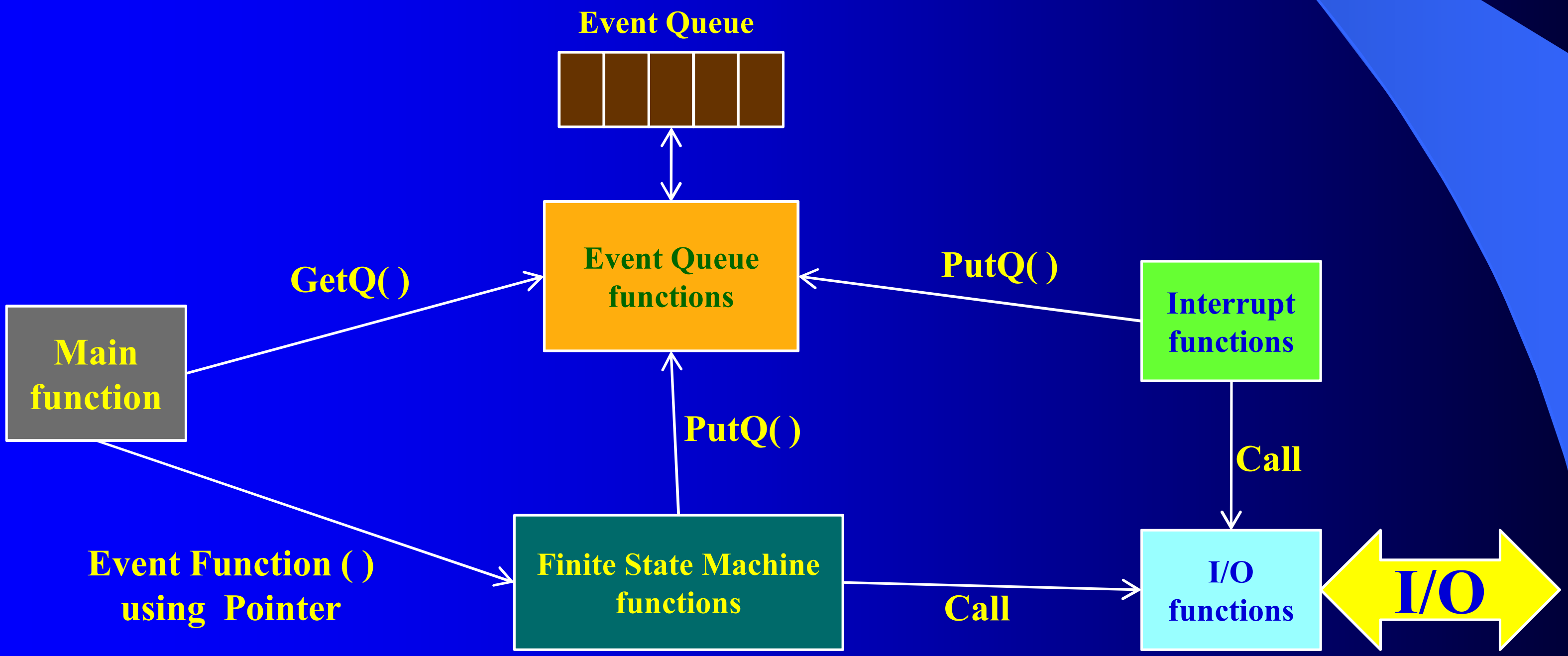
By

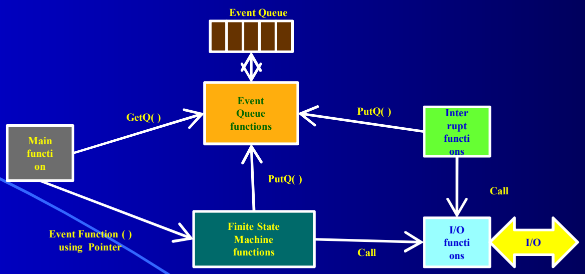
Pornpoj Hanhaboon



การเขียนโปรแกรมแบบ Background/Foreground Event-driven Finite State Machine สามารถแบ่งโปรแกรมเป็น 5 ส่วน

Main function
Event Queue functions
Finite State Machine functions
Interrupt functions
I/O functions





Main function

Initializes hardware.

Initializes Event Queue.

Initializes State Machine.

Initializes other variables.

Makes the infinite loop to get the Event Structures from the Event Queue and calls the Event-Functions using the Event Function Pointers retrieved from the Event Queue.

Event Queue Functions

The library functions for the Event Queue. The Event Queue may be the Array-based FIFO Circular Queue or the Linked-List-based FIFO or Priority Queue for the Event Structures.

Finite State Machine functions

The Event Functions, the Transition Functions and other program components in the files generated by the TI FSM Design Excel program. Programmers have to modify these files to use with their compilers and write the application-specific codes in the Transition Functions and may call some I/O functions. They may also cause the events and make the Event Structures and put those structures into the Event Queue.

Interrupt functions

Interrupt functions for I/Os that cause the events, they should make the Event Structures and put those structures into the Event Queue. They may also call some I/O functions.

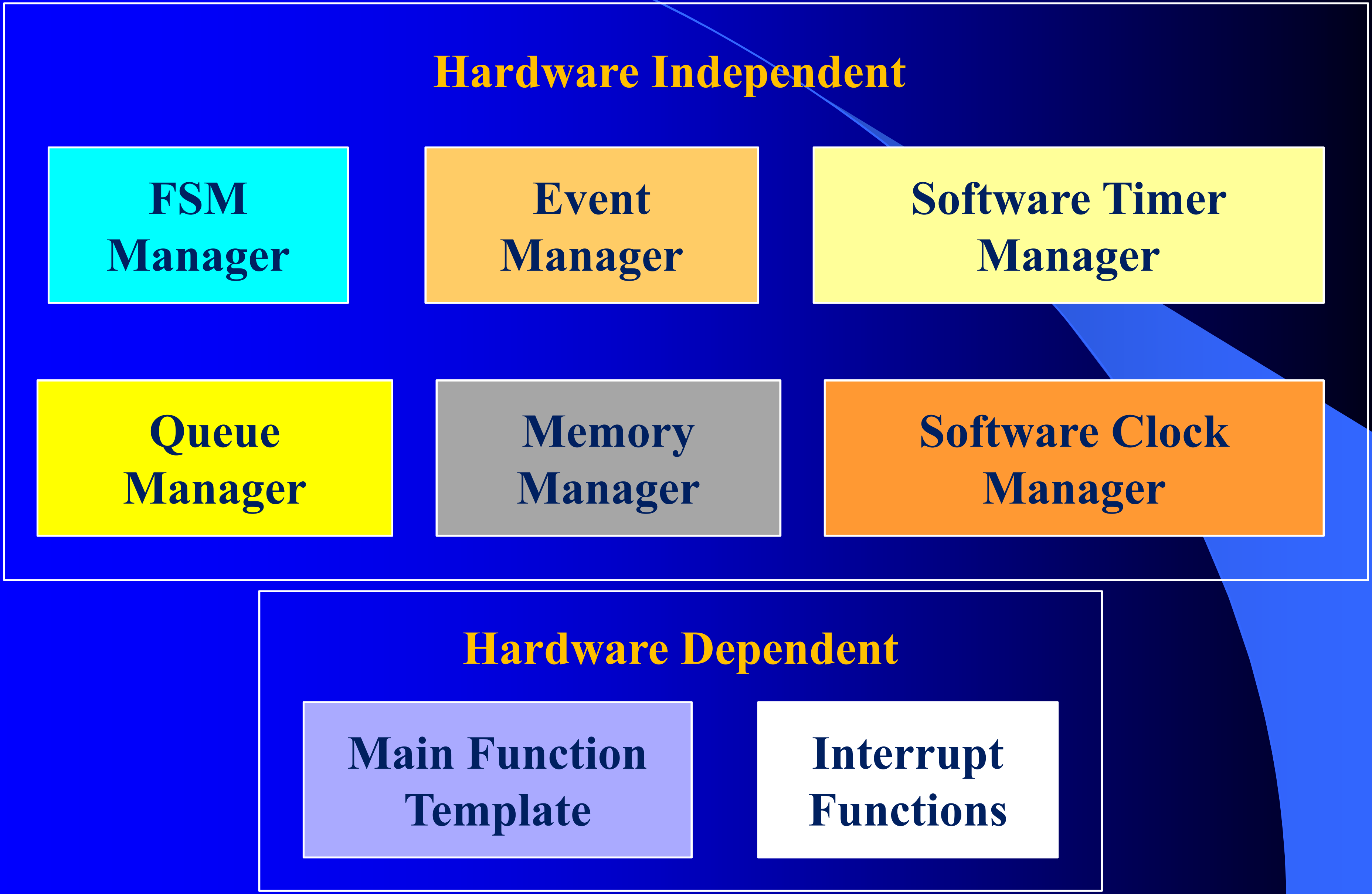
I/O functions

The functions for I/O interfaces, they are hardware specific codes and called by the FSM functions and the interrupt functions.

System-level Data Structures for Event-driven Finite State Machine

Event Control Block Structure
FSM Control Block Structure
Software Timer Control Block Structure
Array-based FIFO Circular Event Queue Structure

Priority-Based Event-driven Multi-FSM Library



Hardware Independent

FSM Manager

Event Manager

Software Timer Manager

Queue Manager

Memory Manager

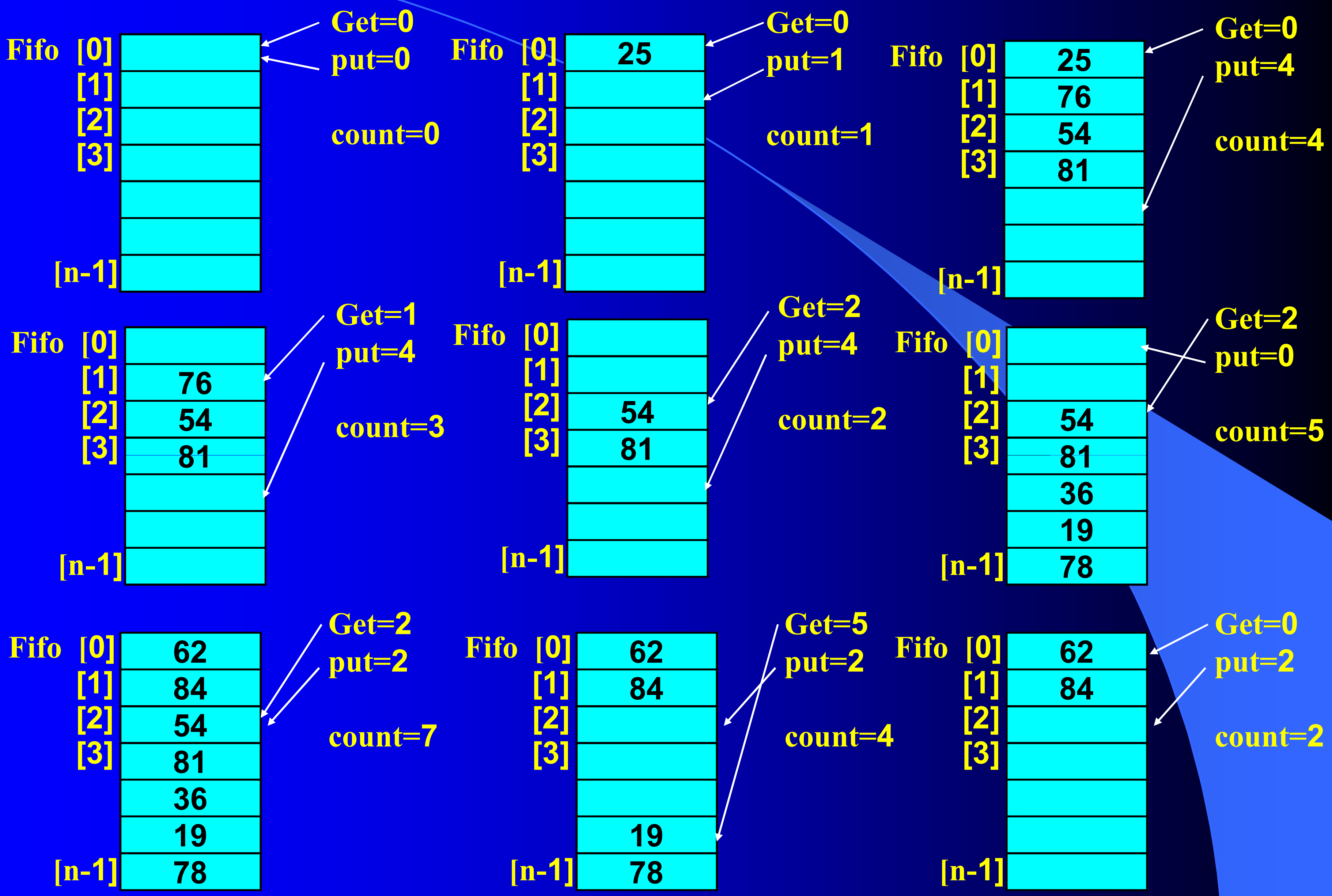
Software Clock Manager

Hardware Dependent

Main Function Template

Interrupt Functions

Array-based FIFO Circular Queue Concept



Fifo	
[0]	
[1]	
[2]	
[3]	
[n-1]	

Get=0
put=0
count=0

Fifo	
[0]	25
[1]	
[2]	
[3]	
[n-1]	

Get=0
put=1
count=1

Fifo	
[0]	25
[1]	76
[2]	54
[3]	81
[n-1]	

Get=0
put=4
count=4

Fifo	
[0]	
[1]	76
[2]	54
[3]	81
[n-1]	

Get=1
put=4
count=3

Fifo	
[0]	
[1]	
[2]	54
[3]	81
[n-1]	

Get=2
put=4
count=2

Fifo	
[0]	
[1]	
[2]	54
[3]	81
	36
	19
[n-1]	78

Get=2
put=0
count=5

Fifo	
[0]	62
[1]	84
[2]	54
[3]	81
	36
	19
[n-1]	78

Get=2
put=2
count=7

Fifo	
[0]	62
[1]	84
[2]	
[3]	
	19
[n-1]	78

Get=5
put=2
count=4

Fifo	
[0]	62
[1]	84
[2]	
[3]	
[n-1]	

Get=0
put=2
count=2

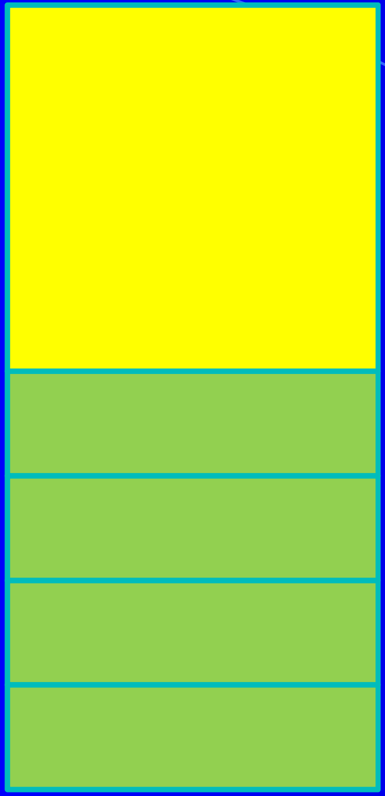
FIFO manager psudocode

```
InitFifo(???)
{
   put = 0;
   get =  0;
    count = 0;
}
```

```
PutFifo(???)
{
    if(count==n)
    {
        ErrHandler();
    }
    else
    {
        count++;
        Fifo[ put ] = data;
        put++;
        if( put== n)
         {
           put = 0;
         }
    }
}
```

```
GetFifo(???)
{
    if( count==0)
    {
       ErrHandler();
    }
    else
    {
        count--;
         *data = Fifo[ get ];
        get++;
        if ( get == n )
         {
           get = 0;
         }
    }
}
```

Q_STRUCT_TYPE_A

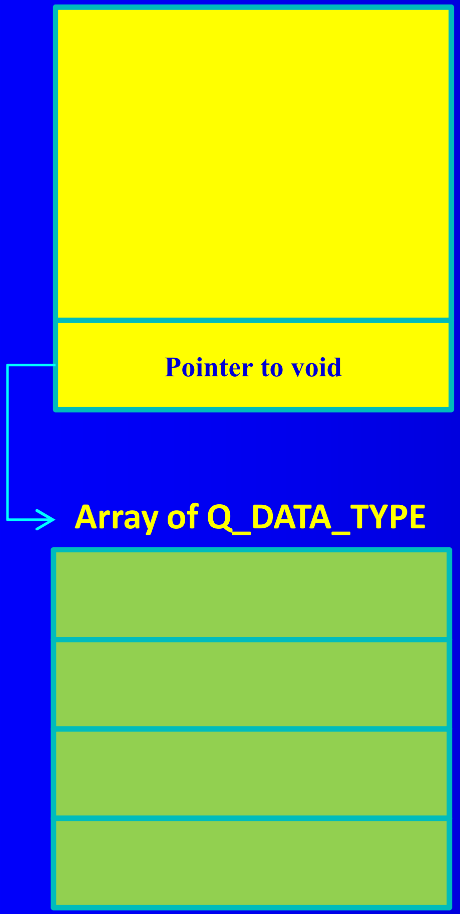


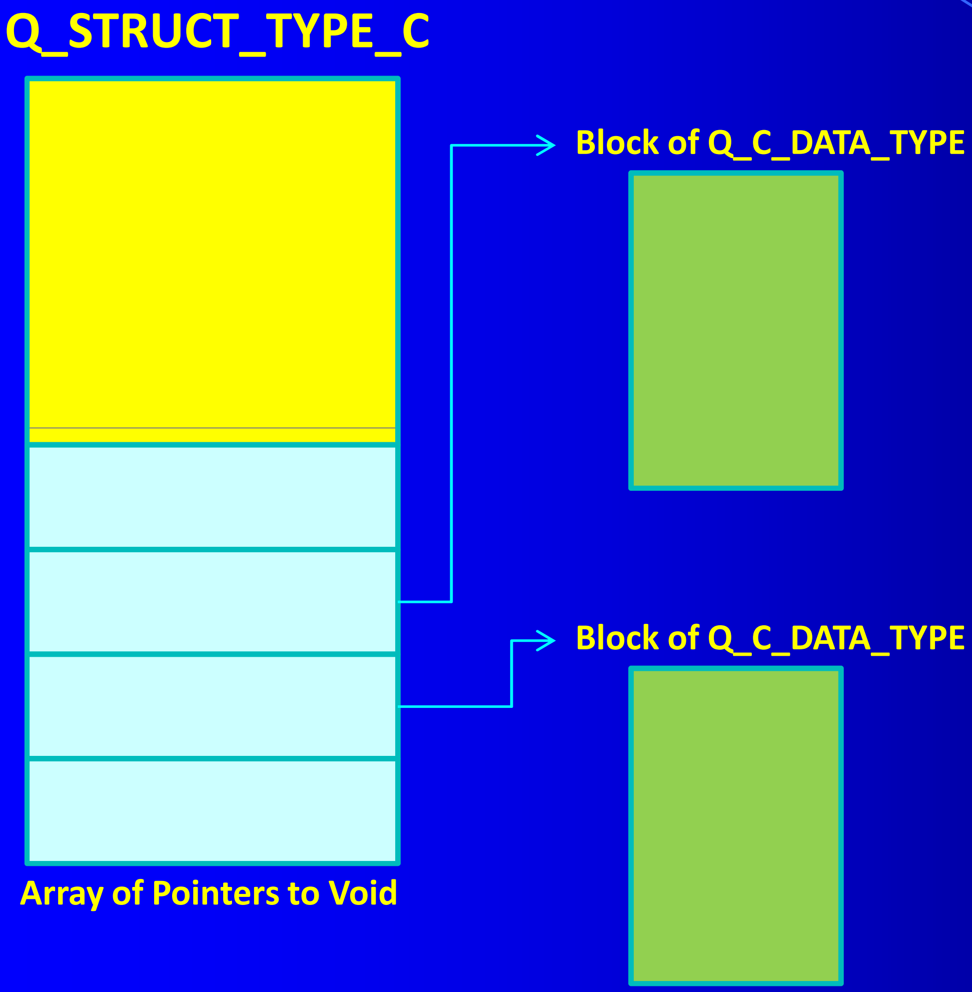
Array of Q_DATA_TYPE

```
#define  Q_ARRAY_LENGTH  ?

typedef   ??????   Q_A_DATA_TYPE;

typedef  struct
{
   intXu itemCount;   //  X  = 8, 16,  32
   intXu indexPut;
   intXu indexGet;
   Q_A_DATA_TYPE  qArray [Q_ARRAY_LENGTH];
}  Q_STRUCT_TYPE_A;
```

```
typedef   ??????   Q_B_DATA_TYPE;

typedef  struct
{
   intXu qLength;     //  X  = 8, 16,  32
   intXu itemCount;
   intXu indexPut;
   intXu indexGet;
   void * qArrayPtr;
}  Q_STRUCT_TYPE_B;
```

Q_STRUCT_TYPE_B





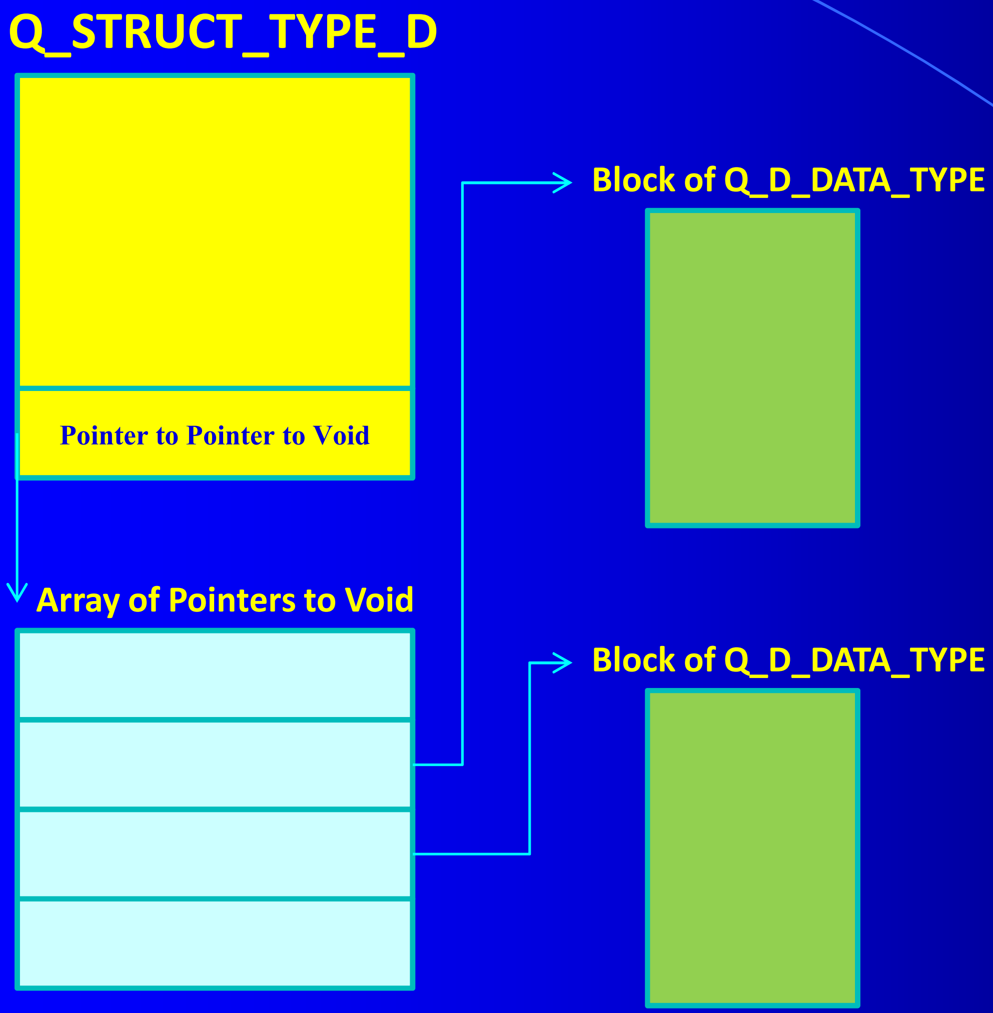
Q_STRUCT_TYPE_C

Block of Q_C_DATA_TYPE

Block of Q_C_DATA_TYPE

Array of Pointers to Void

```
#define  Q_ARRAY_LENGTH  ?

typedef   ??????   Q_C_DATA_TYPE;

typedef  struct
{
   intXu itemCount;   //  X  = 8, 16, 32
   intXu indexPut;
   intXu indexGet;
   void *  qArrayPtr [Q_ARRAY_LENGTH];
} Q_STRUCT_TYPE_C;
```



Q_STRUCT_TYPE_D

Pointer to Pointer to Void

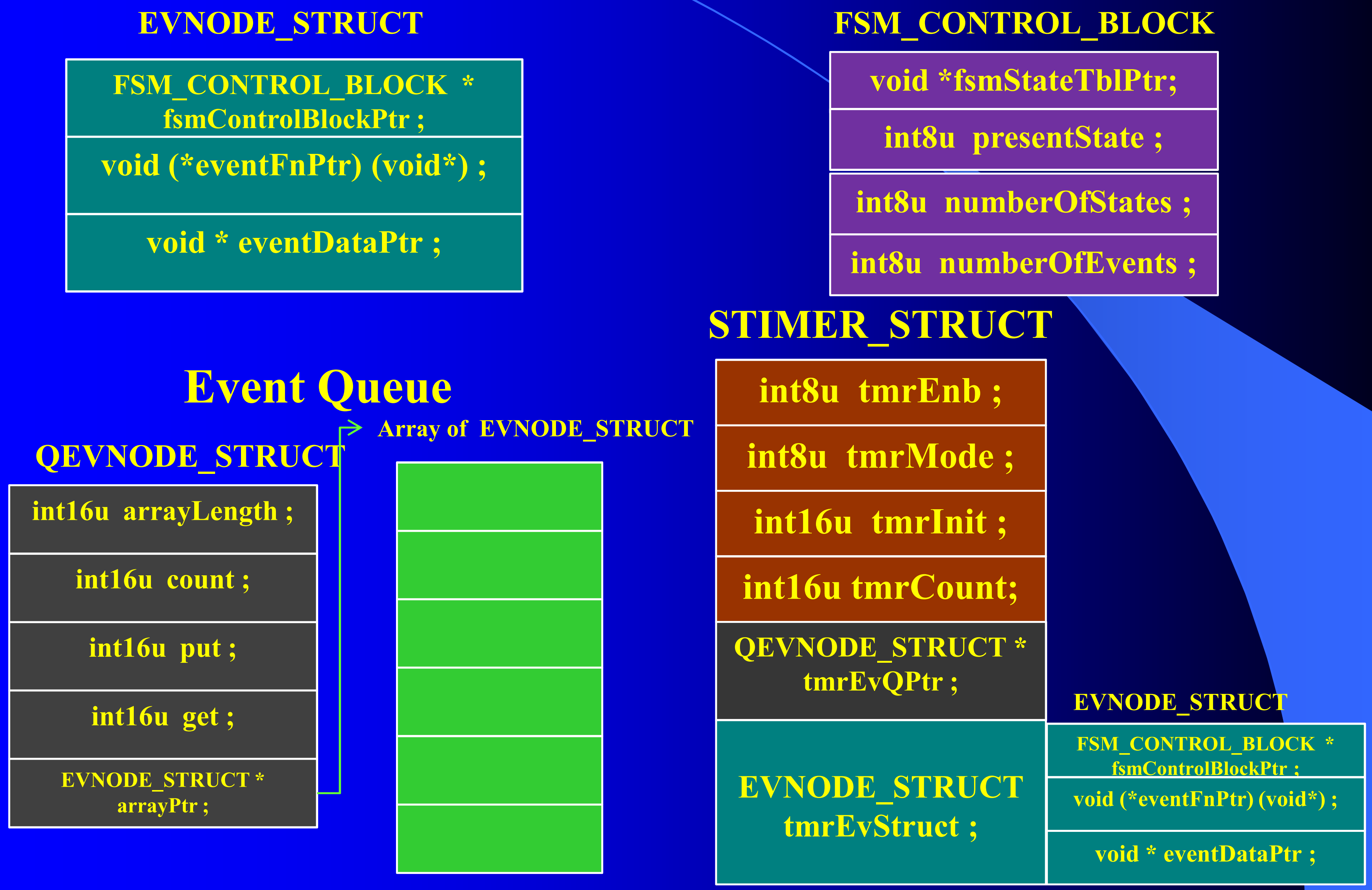
Array of Pointers to Void

Block of Q_D_DATA_TYPE

Block of Q_D_DATA_TYPE

```
typedef  ??????  Q_D_DATA_TYPE;

typedef  struct
{
   intXu qLength;    //  X  = 8, 16,  32
   intXu itemCount;
   intXu indexPut;
   intXu indexGet;
   void ** qArrayPtr;
}  Q_STRUCT_TYPE_D;
```

Background/Foreground Event-driven Finite State Machine Data Structure



EVNODE_STRUCT

FSM_CONTROL_BLOCK * fsmControlBlockPtr ;

void (*eventFnPtr) (void*) ;

void * eventDataPtr ;

FSM_CONTROL_BLOCK

void *fsmStateTblPtr;

int8u presentState ;

int8u numberOfStates ;

int8u numberOfEvents ;

STIMER_STRUCT

int8u tmrEnb ;

int8u tmrMode ;

int16u tmrInit ;

int16u tmrCount;

QEVNODE_STRUCT * tmrEvQPtr ;

EVNODE_STRUCT tmrEvStruct ;

EVNODE_STRUCT

FSM_CONTROL_BLOCK * fsmControlBlockPtr ;

void (*eventFnPtr) (void*) ;

void * eventDataPtr ;

Event Queue

QEVNODE_STRUCT

int16u arrayLength ;

int16u count ;

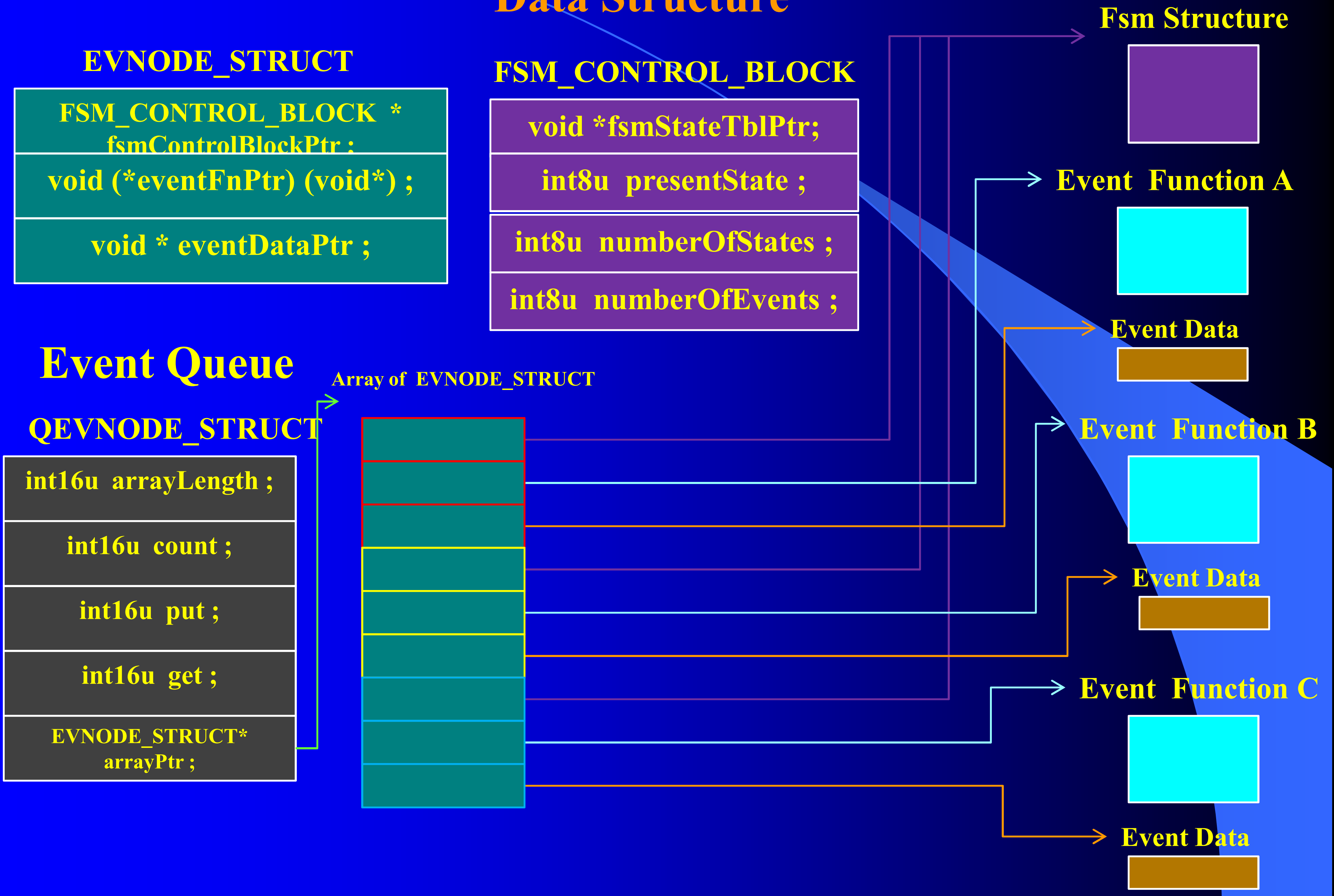
int16u put ;

int16u get ;

EVNODE_STRUCT * arrayPtr ;

Array of EVNODE_STRUCT

Background/Foreground Event-driven Finite State Machine Data Structure



EVNODE_STRUCT

FSM_CONTROL_BLOCK * fsmControlBlockPtr ;

void (*eventFnPtr) (void*) ;

void * eventDataPtr ;

FSM_CONTROL_BLOCK

void *fsmStateTblPtr;

int8u presentState ;

int8u numberOfStates ;

int8u numberOfEvents ;

Event Queue

QEVNODE_STRUCT

int16u arrayLength ;

int16u count ;

int16u put ;

int16u get ;

EVNODE_STRUCT* arrayPtr ;

Array of EVNODE_STRUCT

Fsm Structure

Event Function A

Event Data

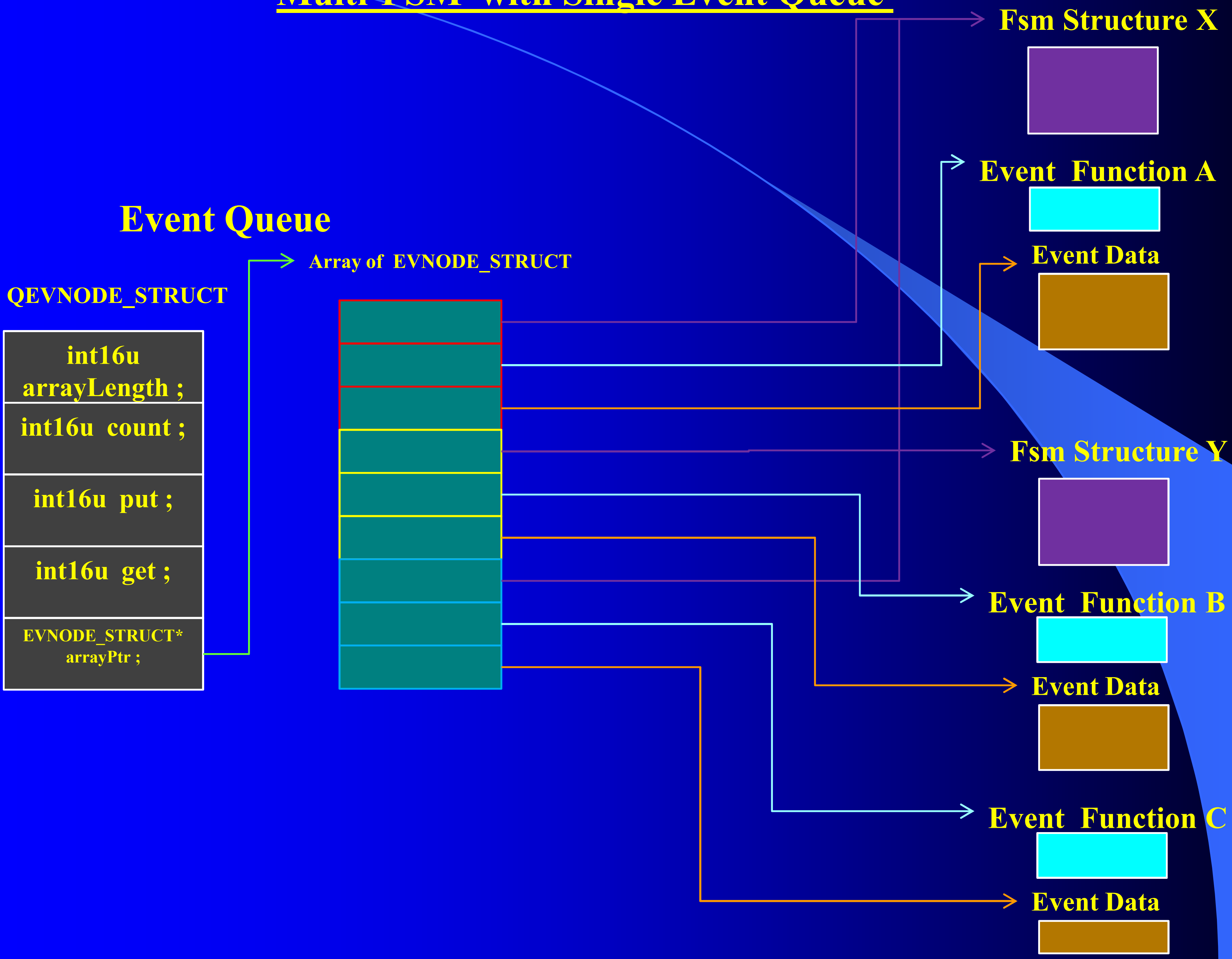
Event Function B

Event Data

Event Function C

Event Data

Multi-FSM with Single Event Queue

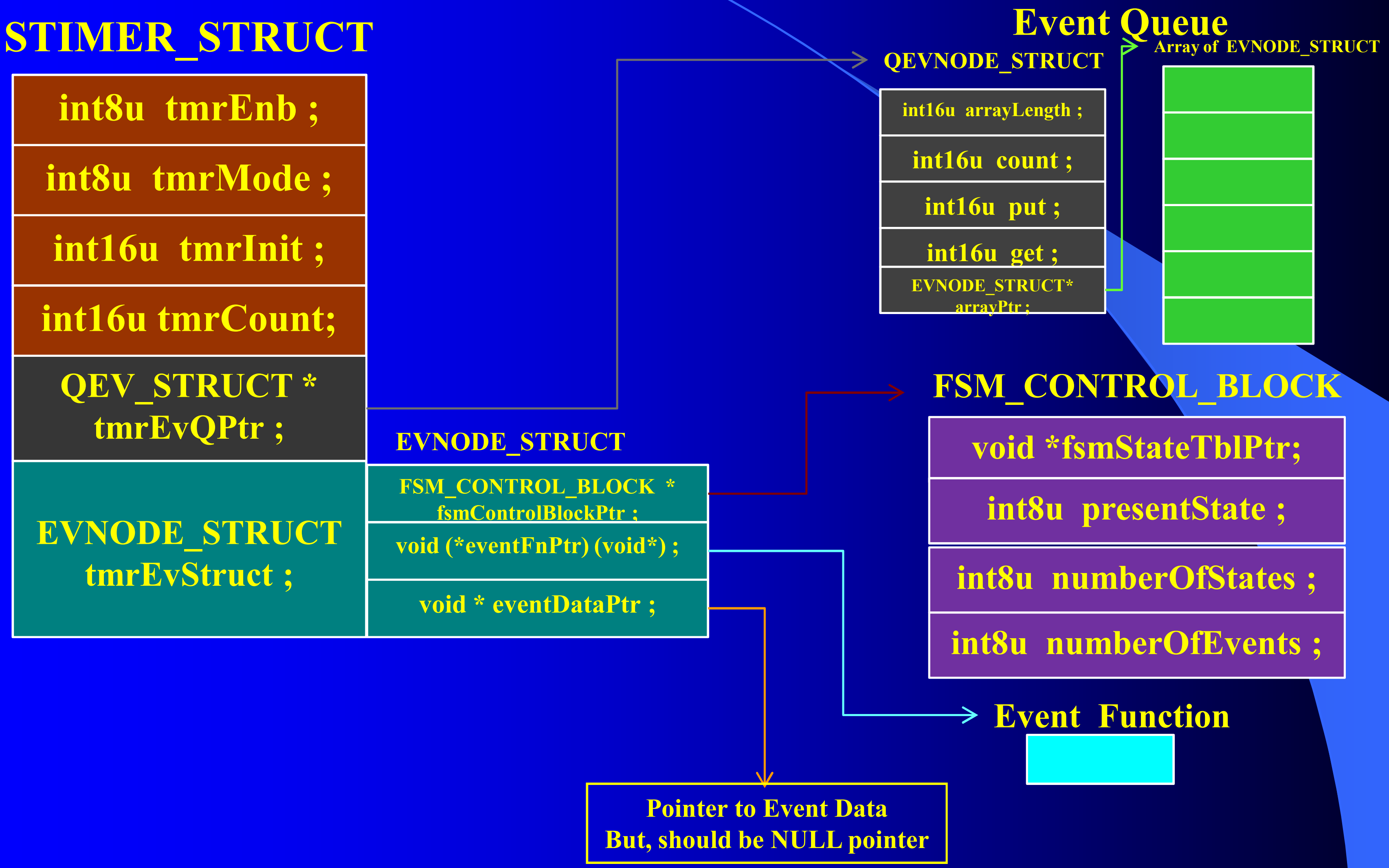


Event Queue

QEVNODE_STRUCT

int16u arrayLength ;
int16u count ;
int16u put ;
int16u get ;
EVNODE_STRUCT* arrayPtr ;

Software Timer Data Structure



STIMER_STRUCT

int8u tmrEnb ;

int8u tmrMode ;

int16u tmrInit ;

int16u tmrCount;

QEV_STRUCT * tmrEvQPtr ;

EVNODE_STRUCT tmrEvStruct ;

EVNODE_STRUCT

FSM_CONTROL_BLOCK * fsmControlBlockPtr ;

void (*eventFnPtr) (void*) ;

void * eventDataPtr ;

Event Queue

QEVNODE_STRUCT

int16u arrayLength ;

int16u count ;

int16u put ;

int16u get ;

EVNODE_STRUCT* arrayPtr ;

Array of EVNODE_STRUCT

FSM_CONTROL_BLOCK

void *fsmStateTblPtr;

int8u presentState ;

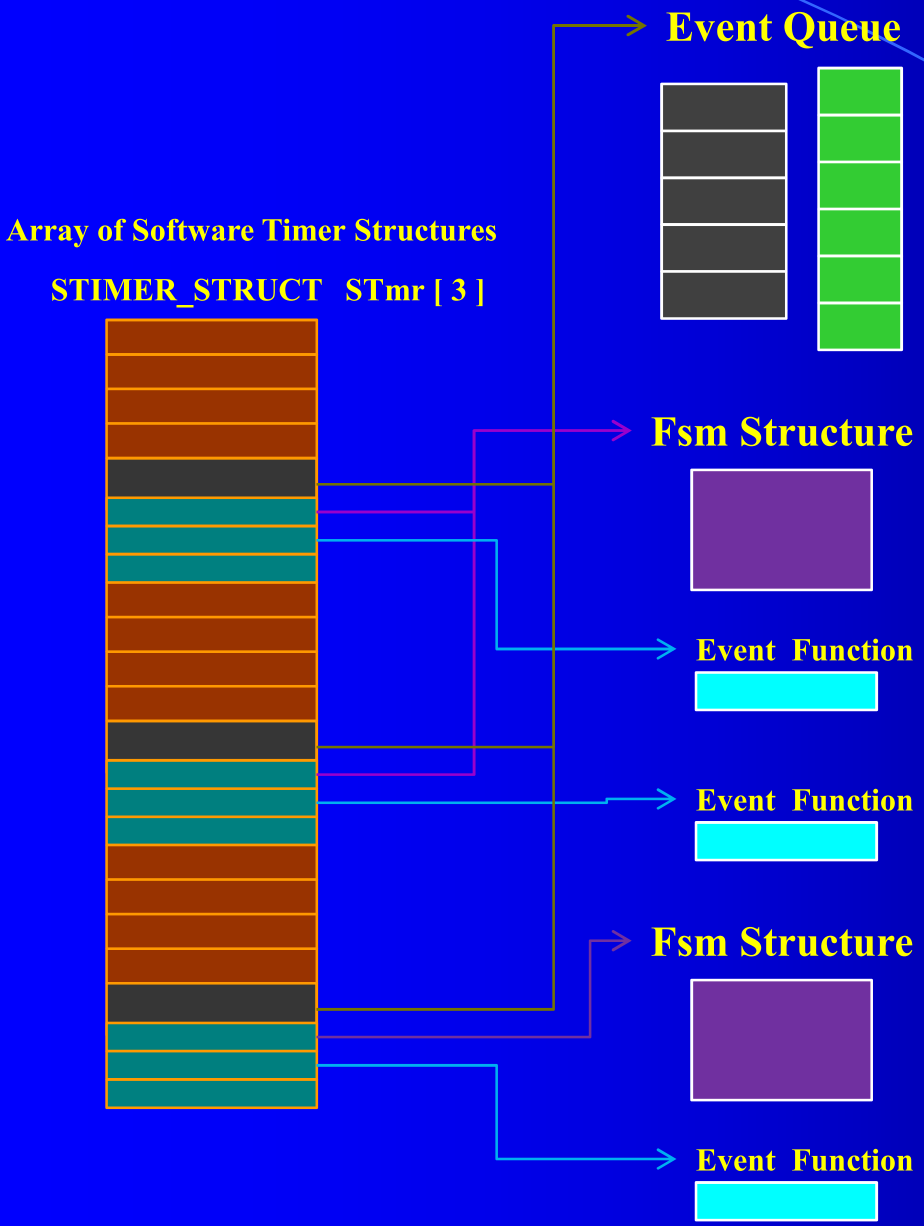
int8u numberOfStates ;

int8u numberOfEvents ;

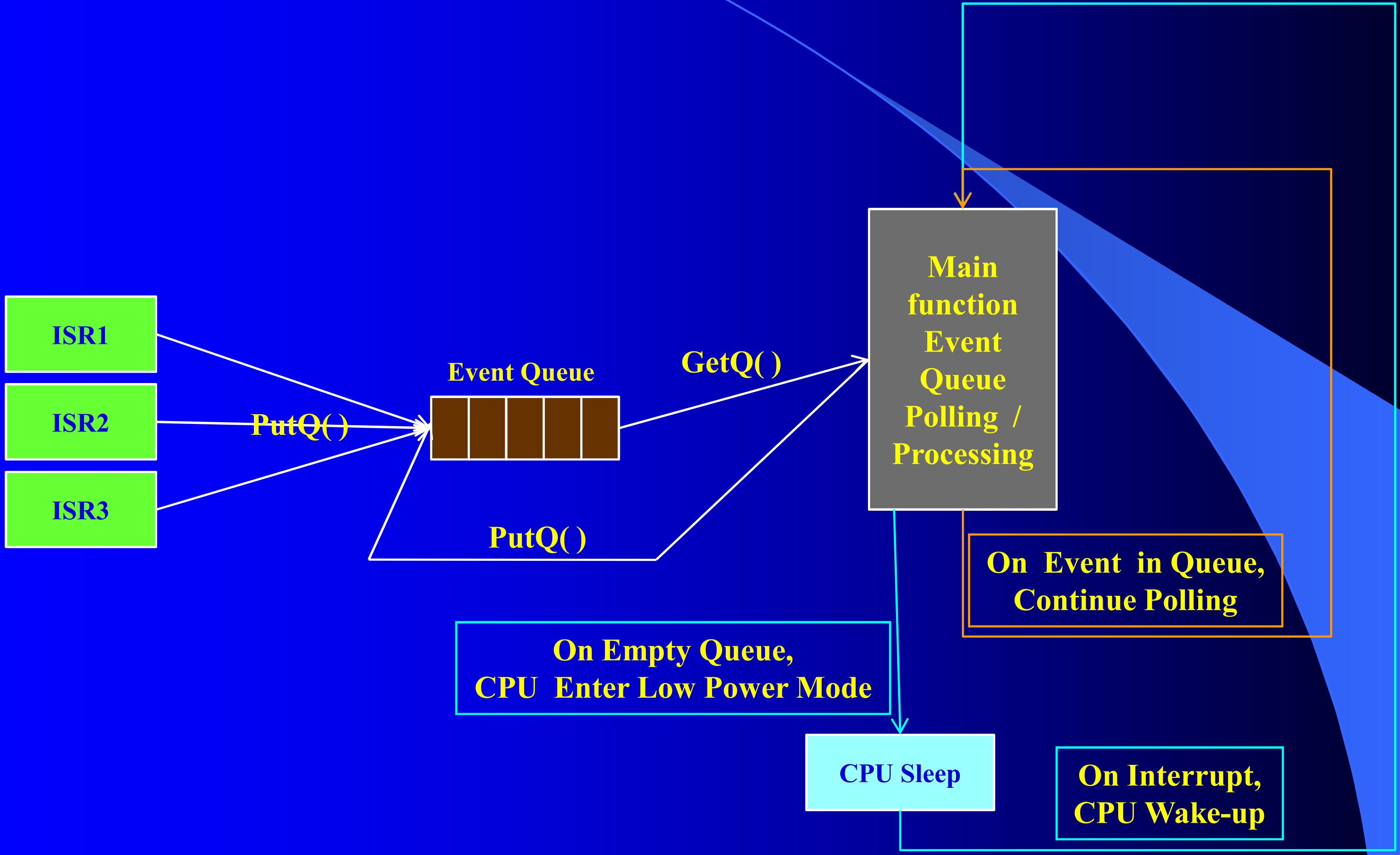
Event Function

Pointer to Event Data
But, should be NULL pointer

Multi-FSM with Single Event Queue and Software Timer

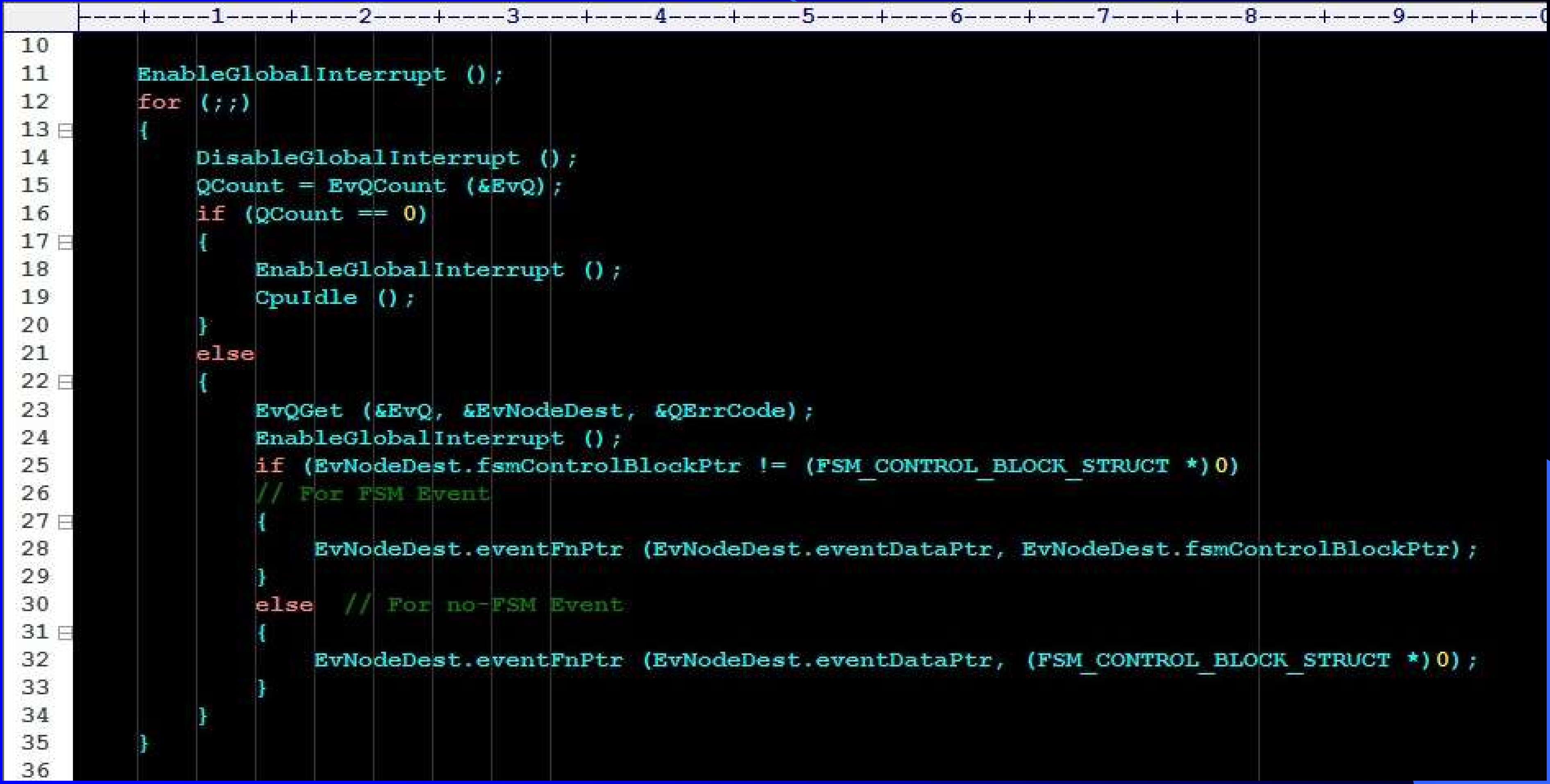


Simple Event-driven Program Structure



ISR1
ISR2
ISR3
PutQ( )
Event Queue
GetQ( )
Main function Event Queue Polling / Processing
PutQ( )
On Event in Queue, Continue Polling
On Empty Queue, CPU Enter Low Power Mode
CPU Sleep
On Interrupt, CPU Wake-up

Infinite Loop for Event-driven with Single Event Queue

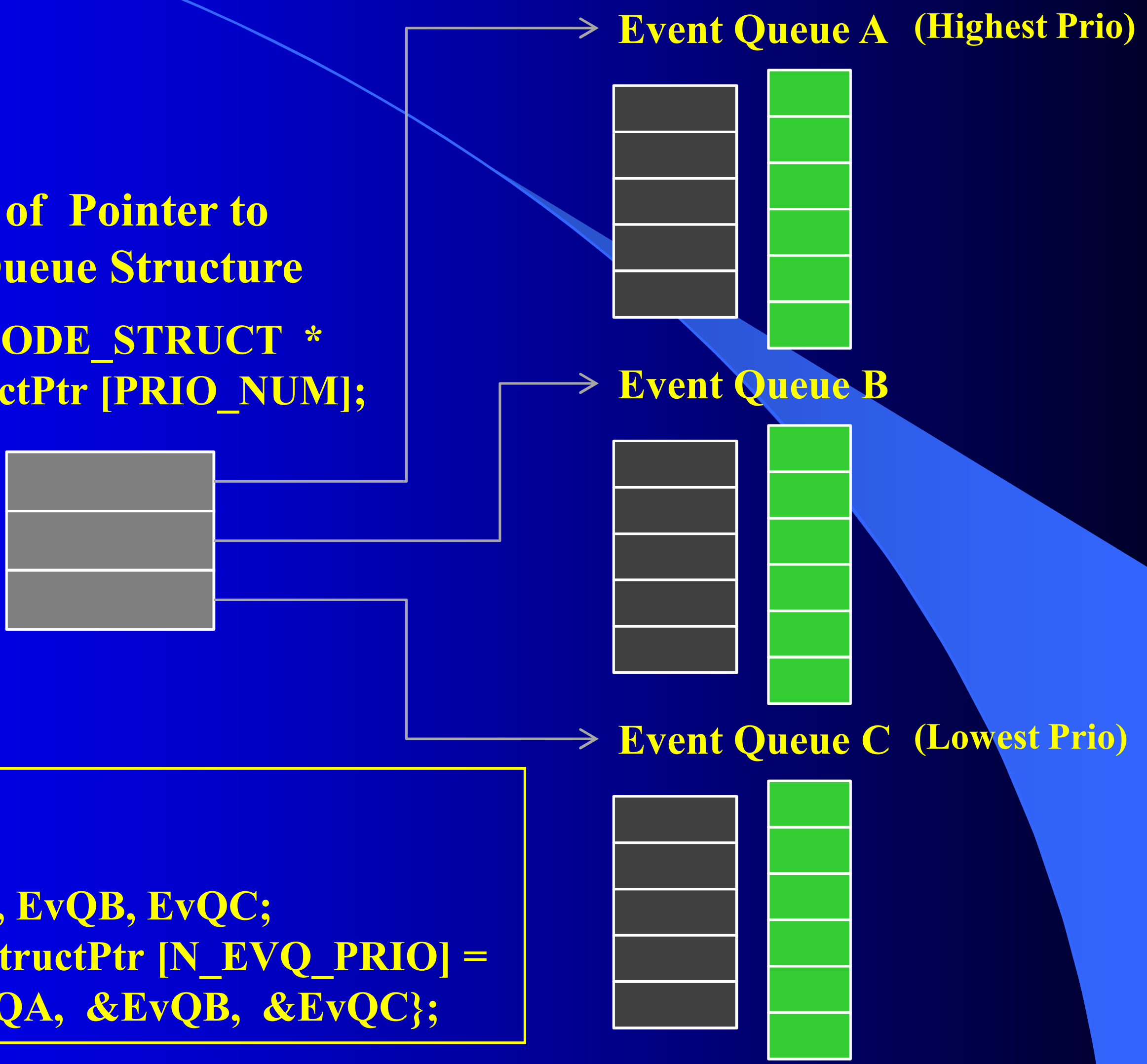


```

    EnableGlobalInterrupt ();
    for (;;)
    {
        DisableGlobalInterrupt ();
        QCount = EvQCount (&EvQ);
        if (QCount == 0)
        {
            EnableGlobalInterrupt ();
            CpuIdle ();
        }
        else
        {
            EvQGet (&EvQ, &EvNodeDest, &QErrCode);
            EnableGlobalInterrupt ();
            if (EvNodeDest.fsmControlBlockPtr != (FSM_CONTROL_BLOCK_STRUCT *)0)
            // For FSM Event
            {
                EvNodeDest.eventFnPtr (EvNodeDest.eventDataPtr, EvNodeDest.fsmControlBlockPtr);
            }
            else  // For no-FSM Event
            {
                EvNodeDest.eventFnPtr (EvNodeDest.eventDataPtr, (FSM_CONTROL_BLOCK_STRUCT *)0);
            }
        }
    }

```

Multi-FSM with Multi-Level Priority Event Queues

Array of Pointer to
Event Queue Structure

QEVNODE_STRUCT *
EvQStructPtr [PRIO_NUM];



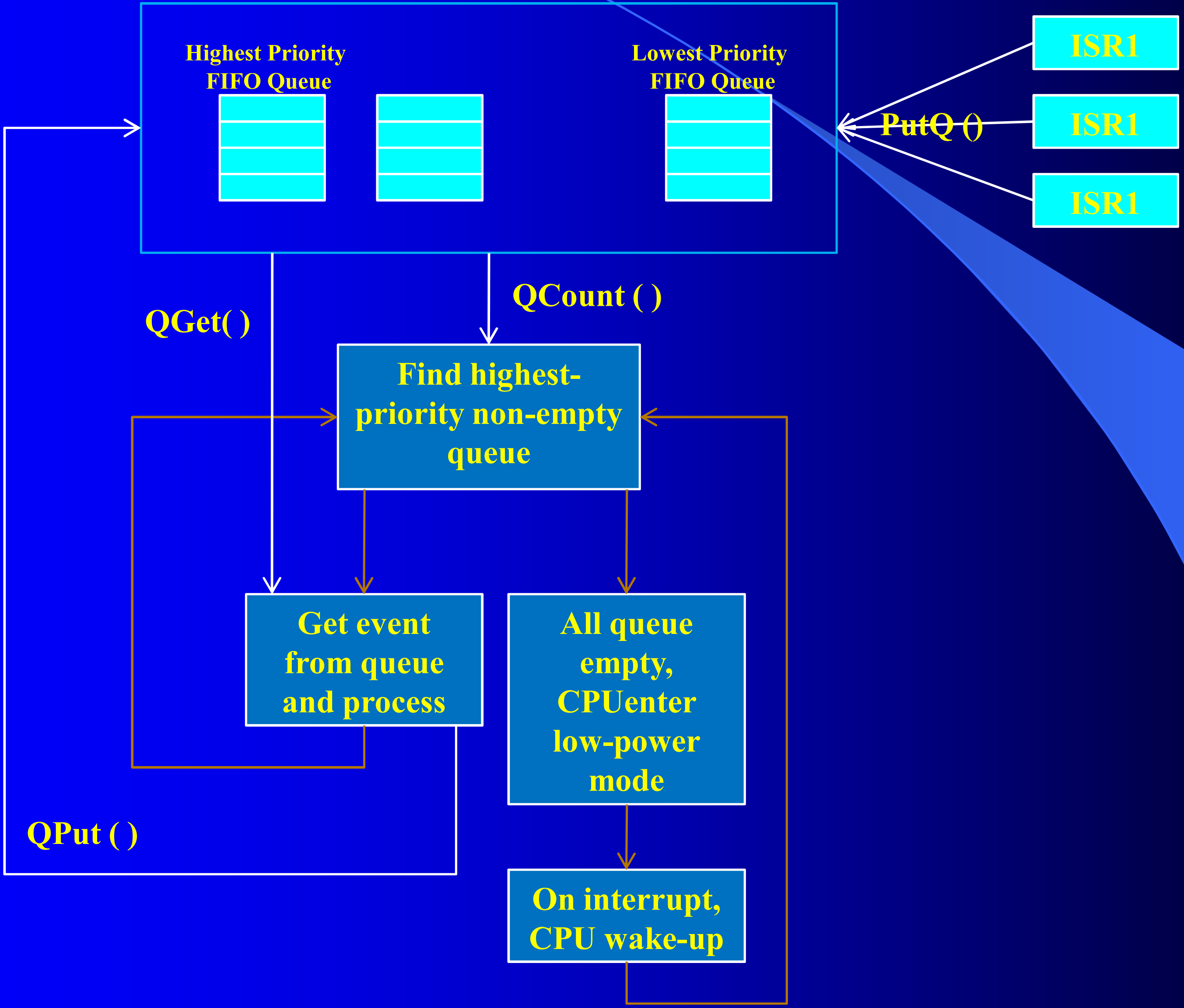
Event Queue A (Highest Prio)

Event Queue B

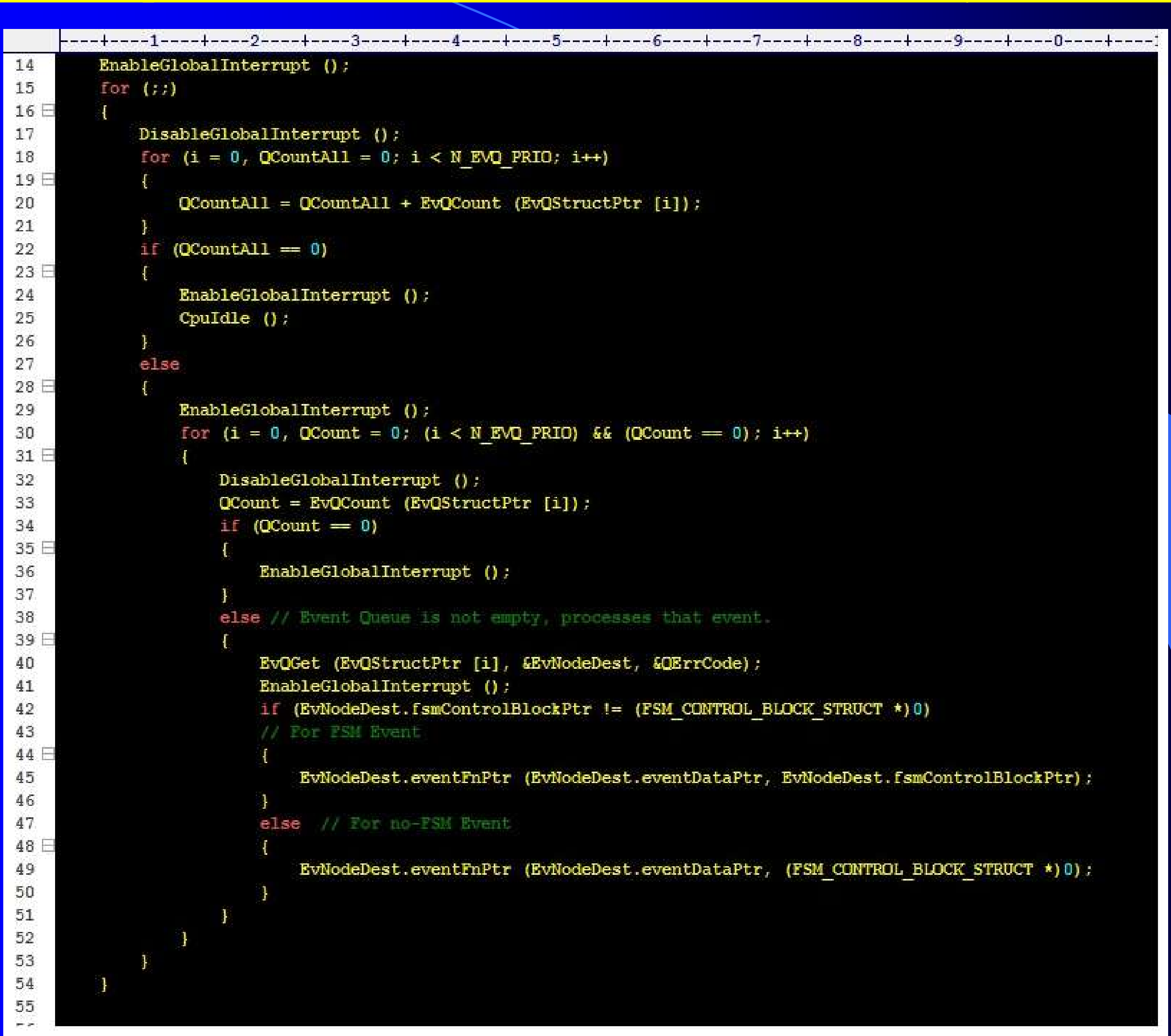
Event Queue C (Lowest Prio)

```
// Example code
#define N_EVQ_PRIO 3
QEVNODE_STRUCT EvQA, EvQB, EvQC;
QEVNODE_STRUCT *EvQStructPtr [N_EVQ_PRIO] =
                    {&EvQA, &EvQB, &EvQC};
```

Event-driven with Multi-level Priority Event Queues

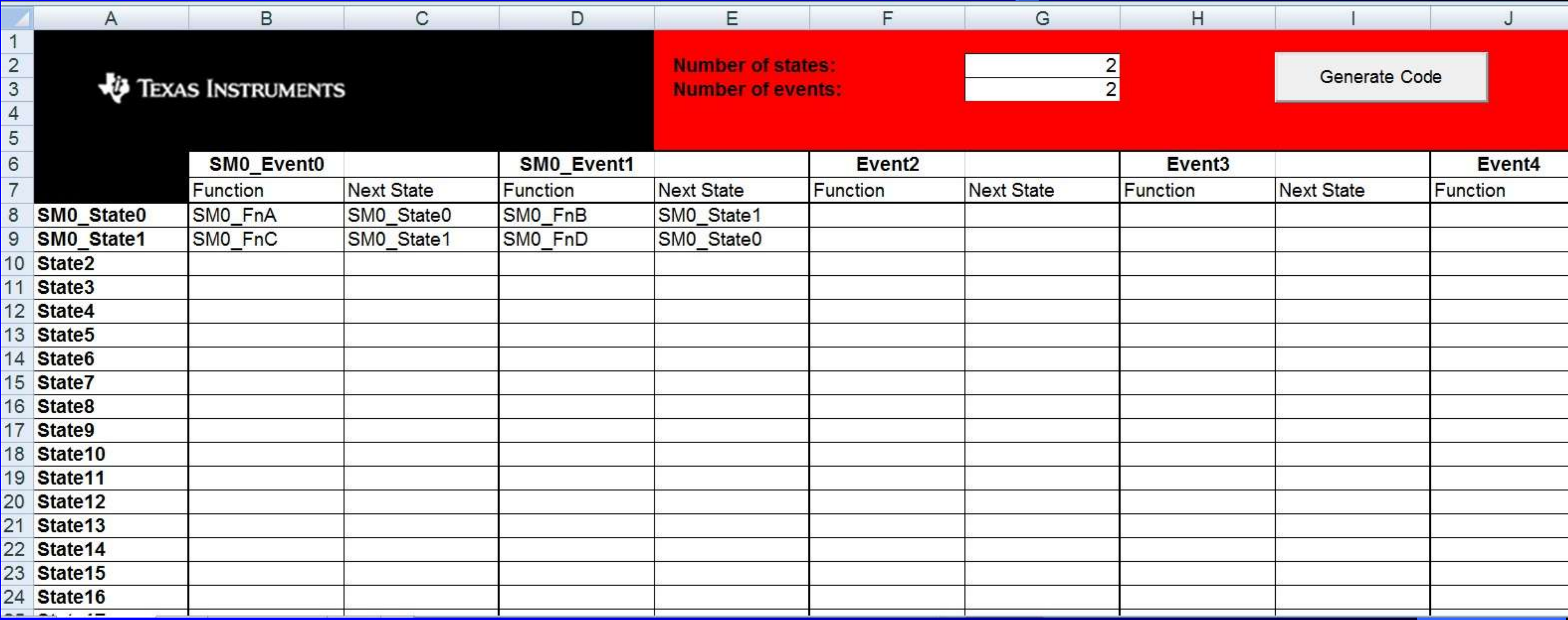

Highest Priority FIFO Queue
Lowest Priority FIFO Queue
ISR1
ISR1
ISR1
PutQ ()
QCount ( )
QGet( )
Find highest-priority non-empty queue
Get event from queue and process
All queue empty, CPUenter low-power mode
On interrupt, CPU wake-up
QPut ( )

Infinite Loop for Event-driven with Multi-level Priority Event Queues

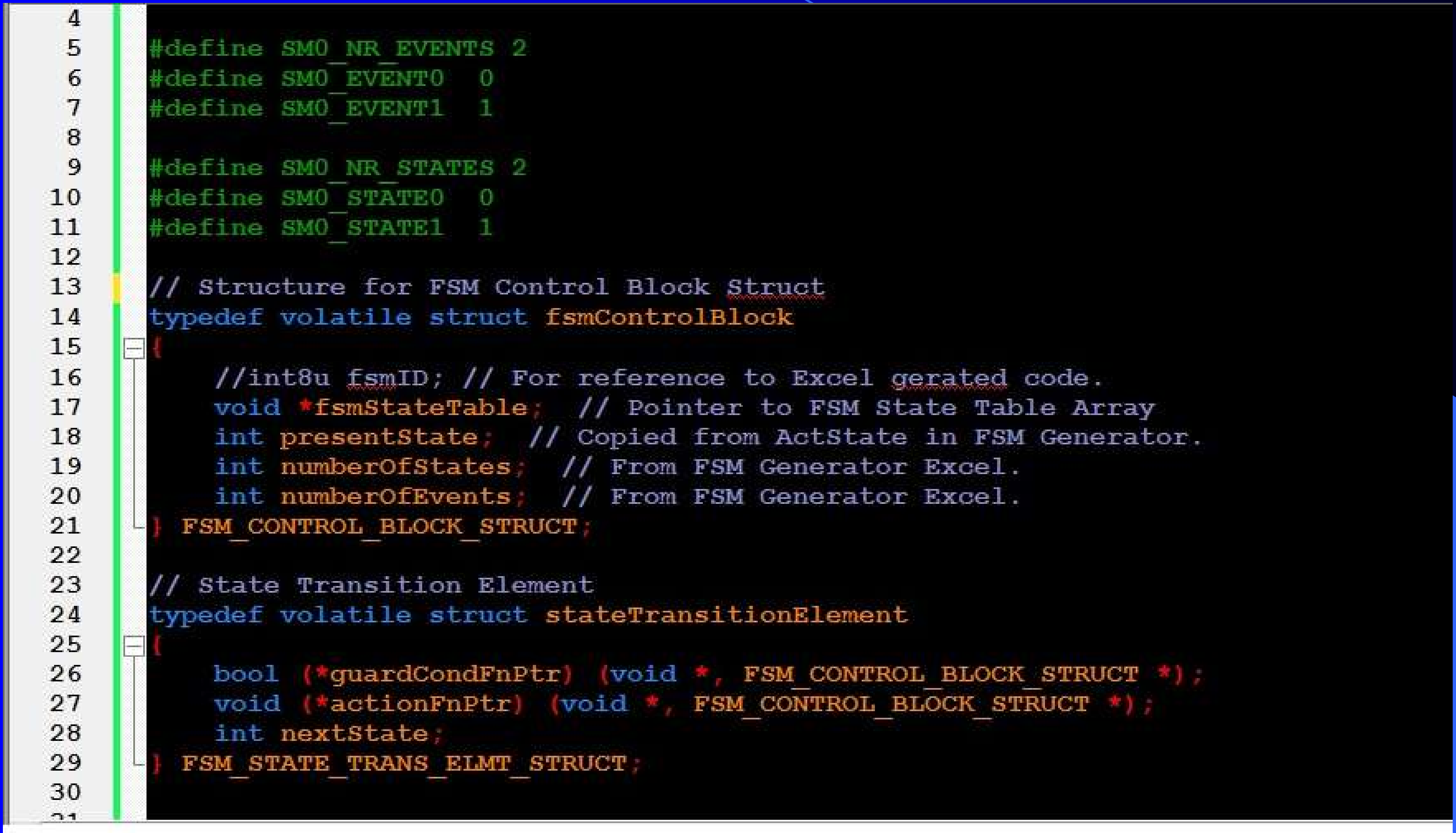


```
    EnableGlobalInterrupt ();
    for (;;)
    {
        DisableGlobalInterrupt ();
        for (i = 0, QCountAll = 0; i < N_EVQ_PRIO; i++)
        {
            QCountAll = QCountAll + EvQCount (EvQStructPtr [i]);
        }
        if (QCountAll == 0)
        {
            EnableGlobalInterrupt ();
            CpuIdle ();
        }
        else
        {
            EnableGlobalInterrupt ();
            for (i = 0, QCount = 0; (i < N_EVQ_PRIO) && (QCount == 0); i++)
            {
                DisableGlobalInterrupt ();
                QCount = EvQCount (EvQStructPtr [i]);
                if (QCount == 0)
                {
                    EnableGlobalInterrupt ();
                }
                else // Event Queue is not empty, processes that event.
                {
                    EvQGet (EvQStructPtr [i], &EvNodeDest, &QErrCode);
                    EnableGlobalInterrupt ();
                    if (EvNodeDest.fsmControlBlockPtr != (FSM_CONTROL_BLOCK_STRUCT *)0)
                    // For FSM Event
                    {
                        EvNodeDest.eventFnPtr (EvNodeDest.eventDataPtr, EvNodeDest.fsmControlBlockPtr);
                    }
                    else  // For no-FSM Event
                    {
                        EvNodeDest.eventFnPtr (EvNodeDest.eventDataPtr, (FSM_CONTROL_BLOCK_STRUCT *)0);
                    }
                }
            }
        }
    }
```

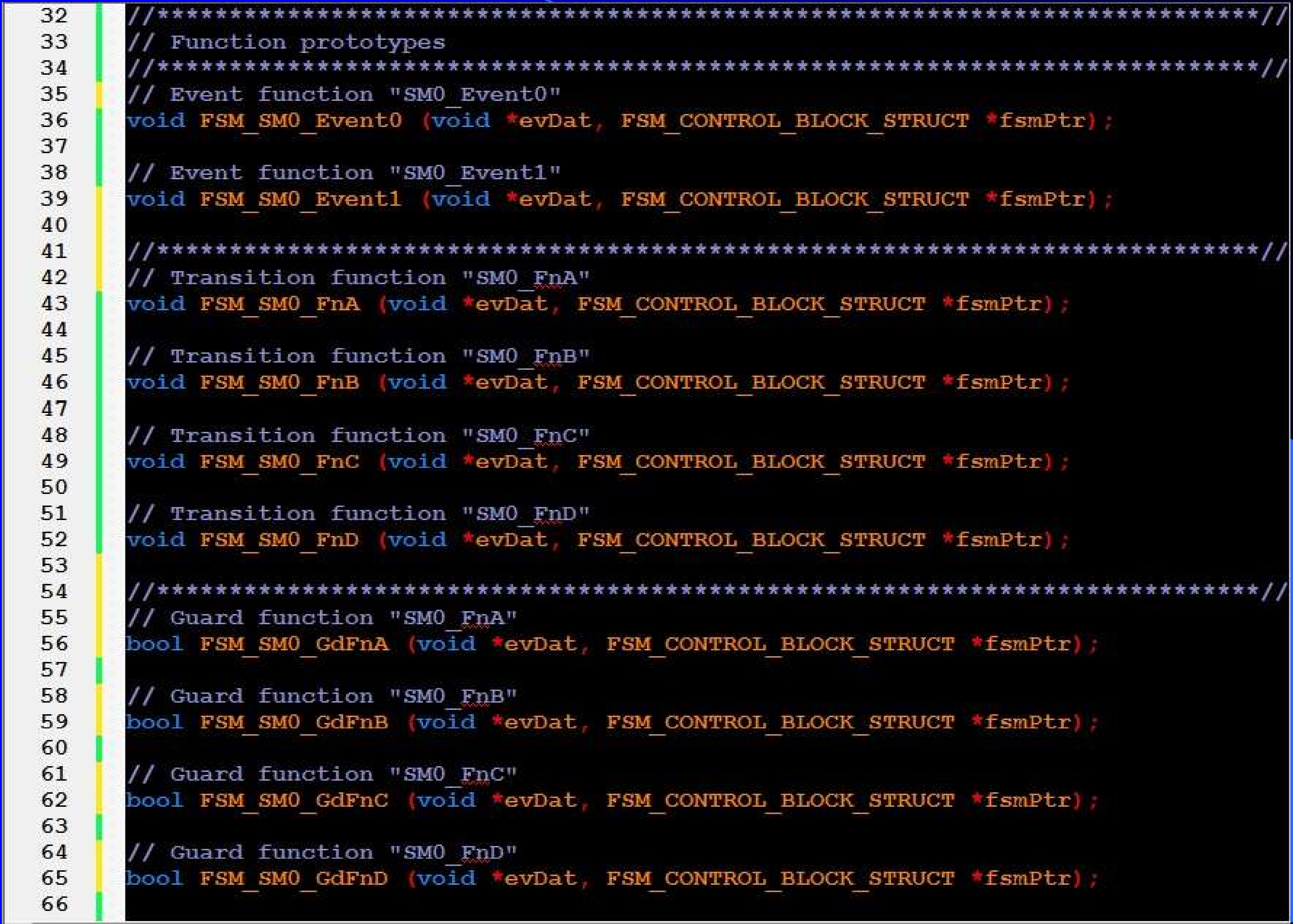
State-Table-Based FSM Programming Example



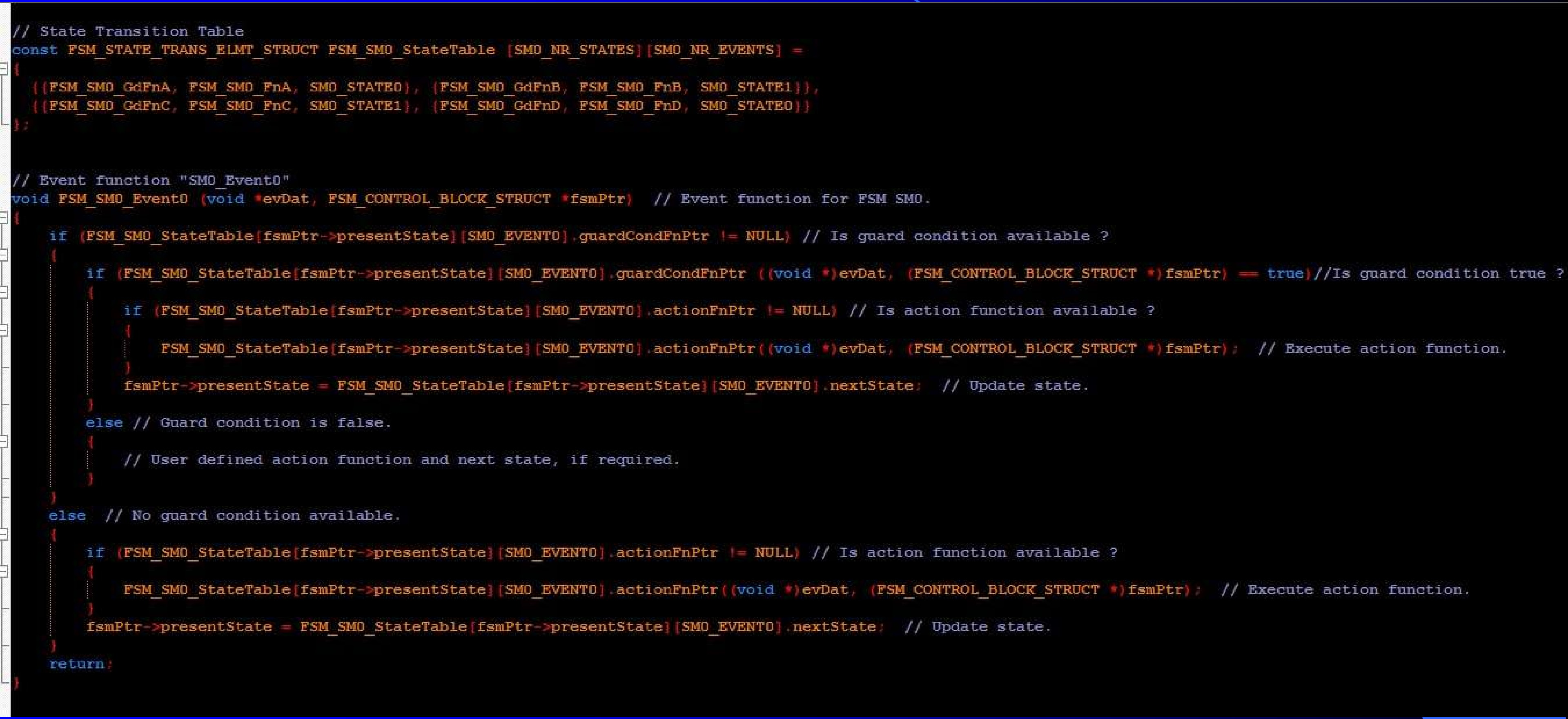
	A	B	C	D	E	F	G	H	I	J
1										
2					Number of states:		2		Generate Code	
3	Texas Instruments				Number of events:		2			
4										
5										
6		**SM0_Event0**		**SM0_Event1**		**Event2**		**Event3**		**Event4**
7		Function	Next State	Function	Next State	Function	Next State	Function	Next State	Function
8	**SM0_State0**	SM0_FnA	SM0_State0	SM0_FnB	SM0_State1					
9	**SM0_State1**	SM0_FnC	SM0_State1	SM0_FnD	SM0_State0					
10	**State2**									
11	**State3**									
12	**State4**									
13	**State5**									
14	**State6**									
15	**State7**									
16	**State8**									
17	**State9**									
18	**State10**									
19	**State11**									
20	**State12**									
21	**State13**									
22	**State14**									
23	**State15**									
24	**State16**									



```

#define SM0_NR_EVENTS 2
#define SM0_EVENT0  0
#define SM0_EVENT1  1

#define SM0_NR_STATES 2
#define SM0_STATE0  0
#define SM0_STATE1  1

// Structure for FSM Control Block Struct
typedef volatile struct fsmControlBlock
{
    //int8u fsmID; // For reference to Excel gerated code.
    void *fsmStateTable;  // Pointer to FSM State Table Array
    int presentState;  // Copied from ActState in FSM Generator.
    int numberOfStates;  // From FSM Generator Excel.
    int numberOfEvents;  // From FSM Generator Excel.
} FSM_CONTROL_BLOCK_STRUCT;

// State Transition Element
typedef volatile struct stateTransitionElement
{
    bool (*guardCondFnPtr) (void *, FSM_CONTROL_BLOCK_STRUCT *);
    void (*actionFnPtr) (void *, FSM_CONTROL_BLOCK_STRUCT *);
    int nextState;
} FSM_STATE_TRANS_ELMT_STRUCT;

```



```c
//****************************************************************************//
// Function prototypes
//****************************************************************************//
// Event function "SM0_Event0"
void FSM_SM0_Event0 (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Event function "SM0_Event1"
void FSM_SM0_Event1 (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

//****************************************************************************//
// Transition function "SM0_FnA"
void FSM_SM0_FnA (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Transition function "SM0_FnB"
void FSM_SM0_FnB (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Transition function "SM0_FnC"
void FSM_SM0_FnC (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Transition function "SM0_FnD"
void FSM_SM0_FnD (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

//****************************************************************************//
// Guard function "SM0_FnA"
bool FSM_SM0_GdFnA (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Guard function "SM0_FnB"
bool FSM_SM0_GdFnB (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Guard function "SM0_FnC"
bool FSM_SM0_GdFnC (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);

// Guard function "SM0_FnD"
bool FSM_SM0_GdFnD (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr);
```



```
// State Transition Table
const FSM_STATE_TRANS_ELMT_STRUCT FSM_SM0_StateTable [SM0_NR_STATES][SM0_NR_EVENTS] =
{
  {{FSM_SM0_GdFnA, FSM_SM0_FnA, SM0_STATE0}, {FSM_SM0_GdFnB, FSM_SM0_FnB, SM0_STATE1}},
  {{FSM_SM0_GdFnC, FSM_SM0_FnC, SM0_STATE1}, {FSM_SM0_GdFnD, FSM_SM0_FnD, SM0_STATE0}}
};


// Event function "SM0_Event0"
void FSM_SM0_Event0 (void *evDat, FSM_CONTROL_BLOCK_STRUCT *fsmPtr)  // Event function for FSM SM0.
{
    if (FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].guardCondFnPtr != NULL) // Is guard condition available ?
    {
        if (FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].guardCondFnPtr ((void *)evDat, (FSM_CONTROL_BLOCK_STRUCT *)fsmPtr) == true)//Is guard condition true ?
        {
            if (FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].actionFnPtr != NULL) // Is action function available ?
            {
                FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].actionFnPtr((void *)evDat, (FSM_CONTROL_BLOCK_STRUCT *)fsmPtr);  // Execute action function.
            }
            fsmPtr->presentState = FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].nextState;  // Update state.
        }
        else // Guard condition is false.
        {
            // User defined action function and next state, if required.
        }
    }
    else  // No guard condition available.
    {
        if (FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].actionFnPtr != NULL) // Is action function available ?
        {
            FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].actionFnPtr((void *)evDat, (FSM_CONTROL_BLOCK_STRUCT *)fsmPtr);  // Execute action function.
        }
        fsmPtr->presentState = FSM_SM0_StateTable[fsmPtr->presentState][SM0_EVENT0].nextState;  // Update state.
    }
    return;
}
```