Components for Embedded Programs

Components for Embedded Programs

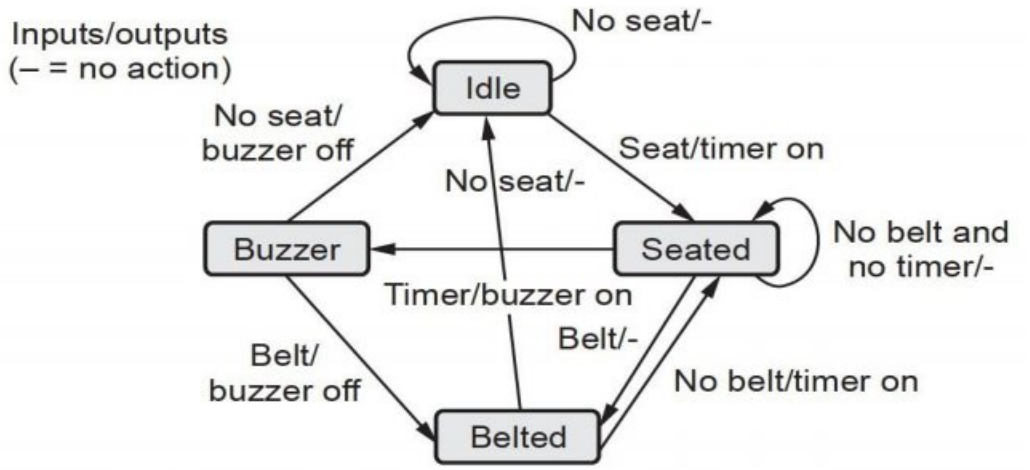
- Embedded software uses three components : State machine, circular buffer and queue.
- State machines are well suited to reactive systems such as user interfaces, circular buffers and queues are useful in digital signal processing.

State Machine

- A state machine is any object that behaves different based on its history and current inputs. Many embedded systems consist of a collection of state machines at various levels of the electronics or software.
- In general, a state machine is any device that stores the status of something at a given time and can operate on input to change the status and/or cause an action or output to take place for any given change.
- In practice, however, state machines are used to develop and describe specific device or program interactions.

1. A set of states.
2. An initial state or record of something stored somewhere.
3. A set of input events.
4. A set of output events.
5. A set of actions or output events that maps the states and input to output.
6. A set of actions or output events that maps the states and inputs to states (which is called a state transition).

- Finite State Automaton (FSA), Finite State Machine (FSM) or State Transition Diagram (STD) is a formal method used in the specification and design of wide range of embedded and real time systems. The system in this case would be represented by a finite number of states.



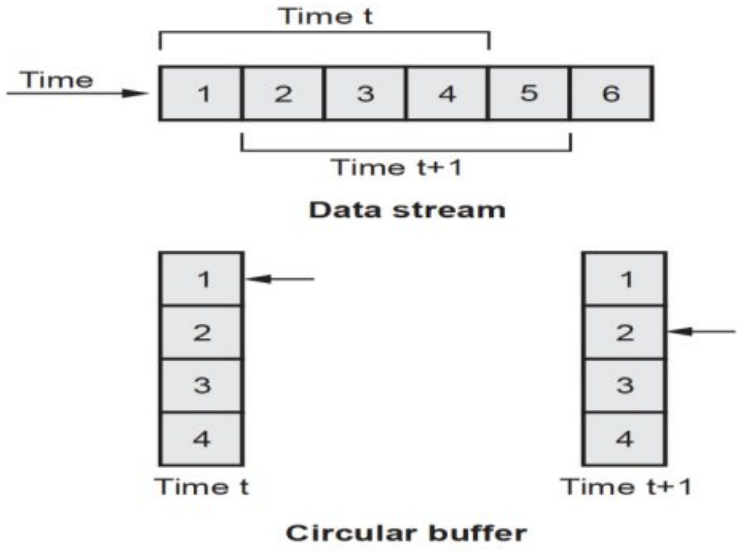
State machine for seat belt controller

- Controllers job is to turn on a buzzer if a person sits in a seat and does not fasten the seat belt within a fixed amount of time. This system has three inputs and one output.
- The inputs are a sensor for the seat to know when a person has sat down, a seat belt sensor that tells when the belt is fastened and a timer that goes off when the required time interval has elapsed. The output is the buzzer.
- The idle state is in force when there is no person in the seat. When the person sits down, the machine goes into the seated state and turns on the timer.
- If the timer goes off before the seat belt is fastened, the machine goes into the buzzer state. If the seat belt goes on first, it enters the belted state. When the person leaves the seat, the machine goes back to idle.

Circular Buffers

- A circular buffer, circular queue, cyclic buffer or ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure lends itself easily to buffering data streams.
- The circular buffer behaviour is ideal for implementing any data structure that is statically allocated and behaves like FIFO.
- Circular buffers are a special type of buffer where the data is circulated around a buffer. In this way they are similar to a single buffer that moves

the next data pointer to the start of the buffer to access the next data. In this way the address pointer circulates around the addresses.



- When a buffer underruns, it indicates that there is no more data in the buffer and that further processing should be stopped. This may indicate an error if the system is designed so that it would never rim out of data.
- If it can happen in normal operation then the data underrun signal indicates a state and not an error. In both cases, a signal is needed to recognise this point.

Queues

- Queues are also used in signal processing and event processing. Queues are used whenever data may arrive and depart at somewhat unpredictable times.
- Queue is also referred to as an elastic buffer. An elastic buffer is a device that helps smooth the data transfer between two similar, but unsynchronized clock domains.
- Linked link is used for building queue.
- For designing the queue, it is declares as follows :

```
# define Q_SIZE 32
#       define Q_MAX (Q_SIZE-1)
int q[Q_SIZE];              /* array for queue */
int head, tail;             /* position of head and tail in the queue */
```

Compilation Techniques

Compilation combines translation and optimization. The high-level language program is translated into the lower-level form of instructions; optimizations try to generate better instruction sequences than would be possible if the brute force technique of independently translating source code statements were used.

1. **Lexical analysis :** The lexical analysis is also called scanning. It is the phase of compilation in which the complete source code is scanned and your source program is broken up into group of strings called token.
2. **Syntax analysis :** The syntax analysis is also called parsing. In this phase the tokens generated by the lexical analyser are grouped together to form a hierarchical structure. The syntax analysis determines the structure of the source string by grouping the tokens together.
3. **Semantic analysis :** Once the syntax is checked in the syntax analyser phase the next phase i.e. the semantic analysis determines the meaning of the source string. For example meaning of source string means matching of parenthesis in the expression, or matching of if ...else statements or performing arithmetic operations of the expressions that are type compatible, or checking the scope of operation.
4. **Intermediate code generation :** The intermediate code is a kind of code which is easy to generate and this code can be easily converted to target code. This code is in variety of forms such as three address code, quadruple, triple, posix.
5. **Code optimization :** The code optimization phase attempts to improve the intermediate code. This is necessary to have a faster executing code or less consumption of memory.
6. **Code generation :** In code generation phase the target code gets generated. The intermediate code instructions are translated into sequence of machine instructions.

Data Structures

Data structure represent the collection of data and various operations that can be performed on this data. There are commonly used data structures such as arrays, linked lists, stacks and so on.

During the code generation for the operations on data structures the compiler must develop memory mapping. For corresponding data elements the suitable memory words are located and then the memory mapping is done.

Arrays are interesting because the address of an array element must in general be computed at run time, since the array index may change

Program Level Energy and Power Analysis and Optimization

Power consumption is a particularly important design metric for battery-powered systems because the battery has a very limited lifetime.

Power consumed by the CPU is a major part of the total power consumption of a computer system and thus has been the main target of power consumption analysis.

How long the device needs to run and whether the batteries can be recharged, need to be thought out ahead of time. In some systems, replacing a battery in a device can be a big expense. This means the system must be conscious of the amount of power it uses and take appropriate steps to conserve battery life. There are several methods to conserve power in an embedded system, including clock control, power-sensitive processors, low-voltage ICs and circuit shutdown.

By measuring the current drawn by the processor as it repeatedly executes distinct instructions or distinct instruction sequences, it is possible to obtain most of the information that is required to evaluate the power consumption of a program for the processor under test.

Power is modeled as a base cost for each instruction plus the inter-instruction overheads that depend on neighboring instructions. The base cost of an instruction can be considered as the cost associated with the basic processing needed to execute the instruction.

However, when sequences of instructions are considered, certain inter-instruction effects come into play, which are not reflected in the cost computed solely from base cost.

1. **Circuit state :** Switching activity depends on the current inputs and previous circuit state.
2. **Resource constraints :** Resource constraints in the CPU can lead to stalls e.g. pipeline stalls and write buffer stalls.
3. **Cache misses :** Another inter-instruction effect is the effect of cache misses.

As the instruction cache size increases, the energy cost of the software on the CPU declines, but the instruction cache comes to dominate the energy consumption.

If the cache is too small, the program runs slowly and the system consumes a lot of power due to the high cost of main memory accesses.

If the cache is too large, the power consumption is high without a corresponding payoff in performance. At intermediate values, the execution time and power consumption are both good.

Method for improving energy consumption :

1. Try to use registers efficiently
2. Analyze cache behavior to find major cache conflicts
3. Make use of page mode accesses in the memory system whenever possible

Some additional observations about energy optimization as follows :

1. Moderate loop unrolling eliminates some loop control overhead
2. Software pipelining reduces pipeline stalls, thereby reducing the average energy per instruction.
3. Eliminating recursive procedure calls where possible saves power by getting rid of function call overhead.

Analysis and Optimization of Program Size, Program Validation and Testing

Analysis and Optimization of Program Size

Data provide an excellent opportunity for minimizing size because the data are most highly dependent on programming style.

In data dominated applications, such as image or speech signal processing applications, summing up the sizes of all the arrays is the most straightforward way to get an upper bound of the memory requirement.

In the data dependency relations in the code are used to find the number of array elements produced or consumed by each assignment, fromwhich a memory trace of upper and lower bounding rectangle as a function of time is found.

Care should be taken while designing buffer size. Data can sometimes be packed, such as by storing several flags in a single word and extracting them by using bit-level operations.

A very low-level technique for minimizing data is to reuse values. Data buffers can often be reused at several different points in the program.

Minimizing the size of the instruction text of a program requires a mix of high-level program transformations and careful instruction selection.

Encapsulating functions in subroutines can reduce program size when done carefully.

Program Validation and Testing

Testing is an organized process to verify the behavior, performance, and reliability of a device or system against designed specifications.

Debugging is the process of removing defects ("bugs") in the design phase to ensure that the synthesized design, when manufactured will behave as expected. Testing is a manufacturing step to ensure that the manufactured device is defect free.

Embedded software development uses specialized compilers and development software that offer means for debugging. Developers build application software on more powerful computers and eventually test the application in the target processing environment.

Testing methods are of two type :

1. **Black-box testing :** This method generates tests without looking at the internal structure of the program.
2. **White box testing :** This method generate tests based on the program structure. This method also called as Clear-box testing.

Black Box Testing

Black box testing is also called functional testing. It is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions.

With black box testing, the software tester does not have access to the source code itself. The code is considered to be a "big black box" to the tester who can't see inside the box.

Black-box is based on requirements and functionality, not code.

Random tests form one category of black-box test. Random values are generated with a given distribution.

The expected values are computed independently of the system, and then the test inputs are applied. A large number of tests must be applied for the results to be statistically significant, but the tests are easy to generate.

Using black box testing techniques, testers examine the high-level design and the customer requirements specification to plan the test cases to ensure the code does what it is intended to do.

Functional testing involves ensuring that the functionality specified in the requirement specification works. System testing involves putting the new program in many different environments to ensure the program works in typical

customer environments with various versions and types of operating systems and/or applications.

Advantages :

1. Tests the final behavior of the software.
2. Can be written independent of software design.
3. Can be used to test different implementations with minimal changes.

Disadvantages :

1. Doesn't necessarily know the boundary cases.
2. Can be difficult to cover all portions of software implementation.

White Box Testing

Often called "structural" testing.

Knowing the internal workings of a product, test that all internal operations are performed according to specifications and all internal components have been exercised.

It involves tests that concentrate on close examination of procedural detail. Logical paths through the software are tested.

White box testing focuses on the internal structure of the software code. The white box tester knows what the code looks like and writes test cases by executing methods with certain parameters.

Test cases exercise specific sets of conditions and loops.

A white-box testing technique that focuses exclusively on the validity of loop constructs. Four different classes of loops exist : Simple loops, nested loops, concatenated loops and unstructured loops.

Advantages :

1. Usually helps getting good coverage.
2. Good for ensuring boundary cases and special cases get tested.

Disadvantages :

1. Tests based on design might miss bigger picture system problems.
2. Tests need to be changed if implementation/algorithm changes.
3. Hard to test code that isn't there (missing functionality) with white box testing.