Lucas Seara Manoel

# Relatório 2 - Sistema de Controle Digital modelado no Espaço de Estados

Florianópolis - SC, Brasil
1 de Dezembro de 2018

# Relatório 2 - Sistema de Controle Digital modelado no Espaço de Estados

Lucas Seara Manoel*

1 de Dezembro de 2018

## Sumário



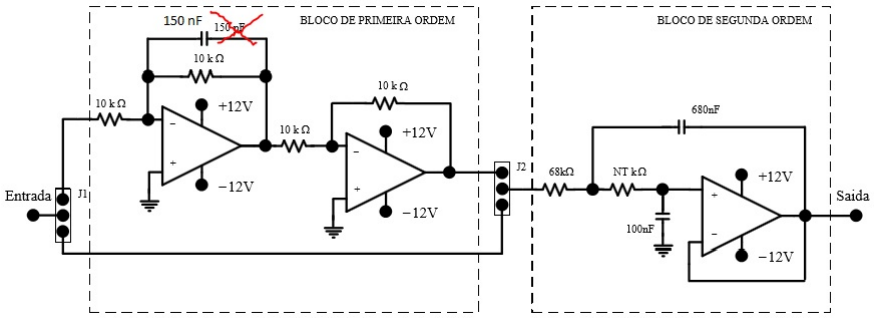
*Aluno de Graduação de Eng. Eletrônica(IFSC/DAELN) Mat:. 131004417-1

# 1 Introdução.

O surgimento de sistemas de processamento digital proporcionaram aos projetistas de sistemas de controle uma alternativa de realizar sistemas de compensação de ordem elevada com baixo custo. Por meio de conversores de sinal analógico para digital e moduladores de sinal por largura de pulso, microprocessadores que estão cada vez mais modernos podem interagir com o mundo analógico de forma a controlar o comportamento de uma planta analógica. Com a discretização de uma planta contínua representado no espaço de estados, é possível controlar o comportamento dinâmico do sistema por meio da realocação de seus polos.

O objetivo da atividade descrita nesse relatório consiste em desenvolver um controlador digital modelado no espaço de estados capaz de reduzir o sobressinal e o tempo de acomodação da resposta ao degrau da planta apresentada na Figura 1 pela metade.

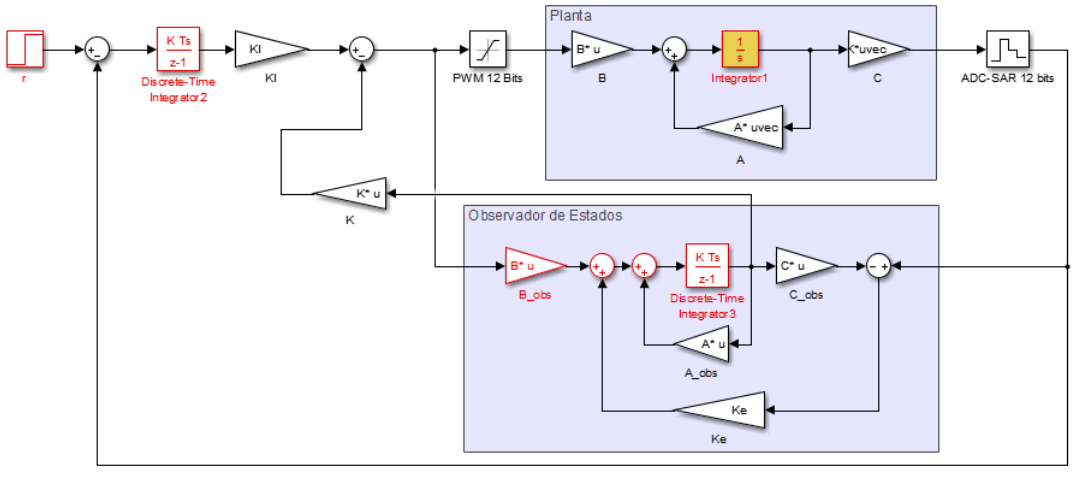
Figura 1 – Esquemático da planta proposta para a atividade.




Fonte: Slide de apresentação do projeto

A Figura 2 ilustra o esquemático da planta apresentada na Figura 1 em conjunto com o sistema de controle formando um sistema em malha fechada. A interface do sistema digital com o sistema analógico, no sentido do fluxo de sinal do sistema digital para o analógico, será feito com modulação de largura de pulso (PWM - Pulse Width Modulation). A interface no sentido do fluxo de sinal do sistema analógico para o sistema digital será feito por meio de conversores analógicos digitais.

Figura 2 – Esquemático do Sistema - Planta e Controlador.



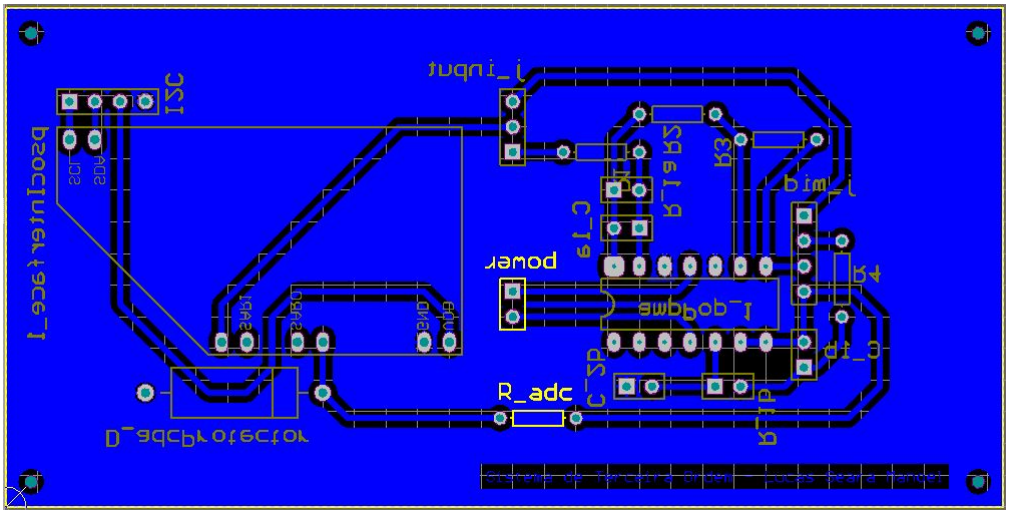

Software: MATLAB - Simulink

# 2 Montagem da planta e sistema de controle.

## 2.1 Montagem da planta em placa de circuito impresso.

O layout da planta foi desenhado por meio do *software* Altium. O amplificador operacional utilizado foi o LM224n. O layout abaixo segue o esquemático da Figura 1 com um diodo zener 1N4734A de 5.6 V para proteção do ADC do microcontrolador contra tensão acima das suportadas pelo pino do microcontrolador (que segundo o *datasheet* é 5.5 V para os microcontroladores da Cypress da linha PSOC 5LP). Essa proteção é pertinente uma vez que os amplificadores operacionais irão operar com tensão simétrica de 12 V.

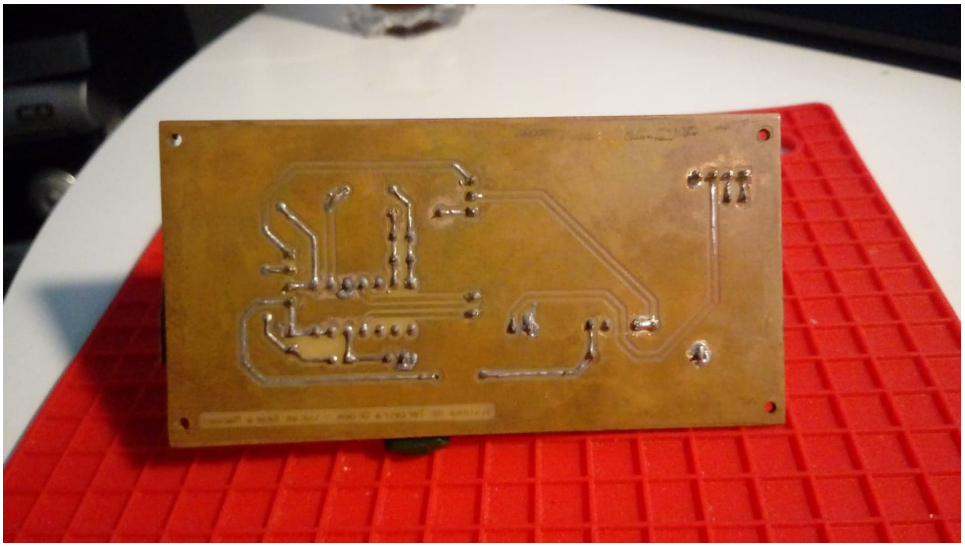
Figura 3 – Layout da placa de circuito impresso.




Software: Altium

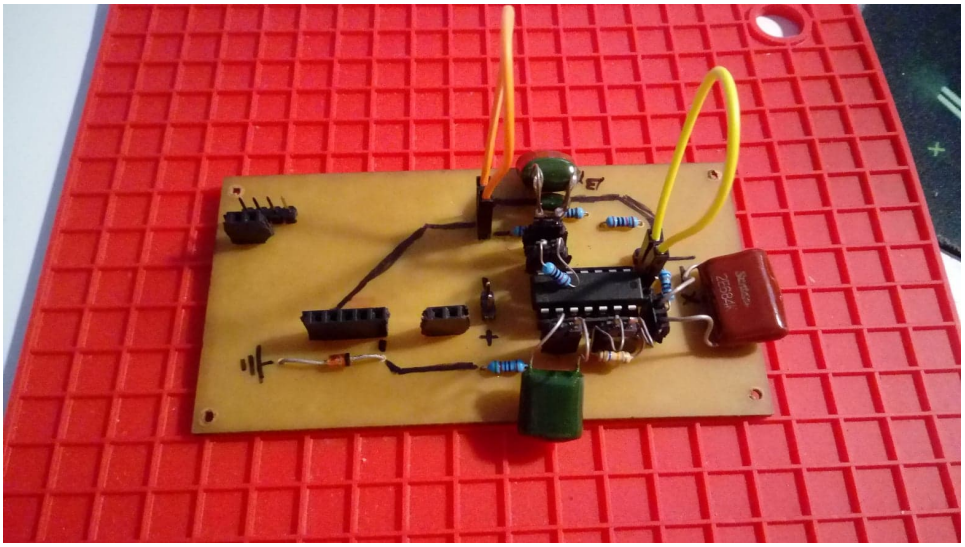
A Figura 4 apresenta como ficou a placa de circuito impresso após corrosão. As dimensões da placa são 5x10 cm:

Figura 4 – Placa de circuito impresso após corrosão.



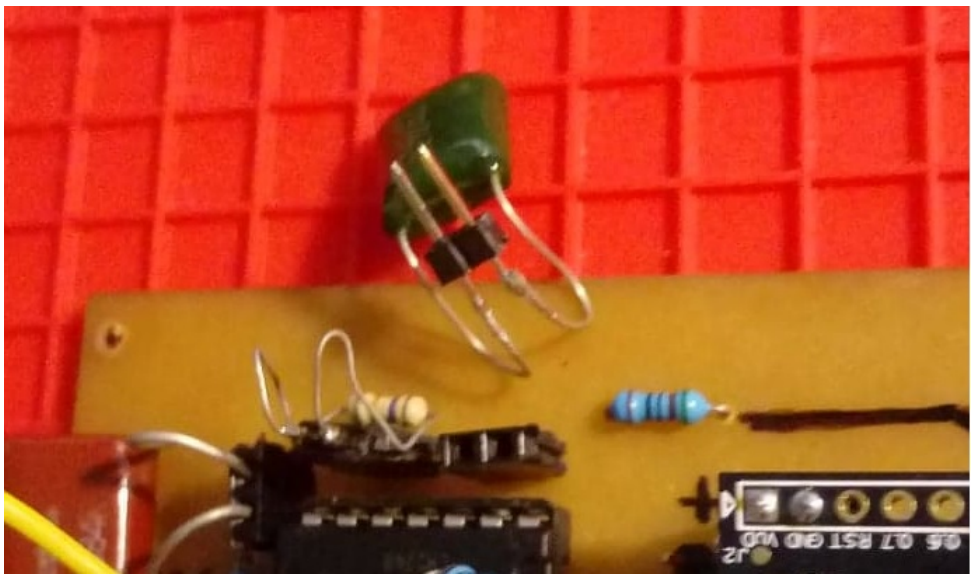
O layout do da planta foi desenhado de forma que os elementos chaves que definem as constantes de tempo possam ser facilmente substituídos. Dessa forma é evitado que seja preciso reaquecer as trilhas de cobre da placa de fenolite em um processo de retrabalho de soldagem. Essa abordagem facilitou a troca de um dos capacitores do sistema de primeira ordem que foi substituído durante o decorrer do projeto (essa troca está indicada com uma marcação X em vermelho na Figura 1 - troca do capacitor de 150 pF para 150 nF).

Figura 5 – Placa de circuito - Top Layer.



Para prevenir o mal contato entre os terminais dos componentes removíveis com a planta, foram soldados aos terminais dos componentes pinos que exercem um bom encaixe com os *headers* da planta.

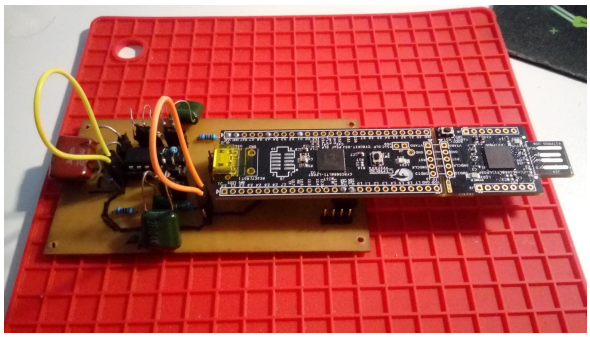
Figura 6 – Placa de circuito impresso - Componentes Removíveis.



## 2.2 Implementação do sistema de controle com uso do dispositivo PSOC.

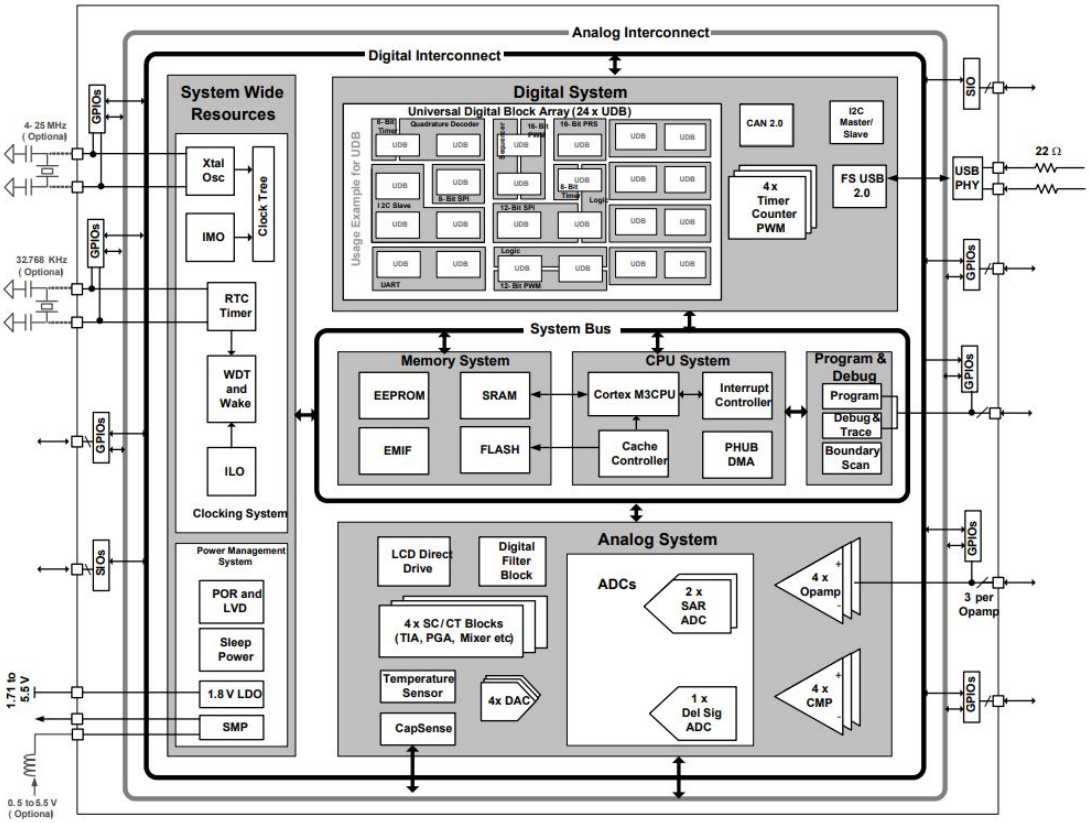
O layout foi desenvolvido com base na placa de desenvolvimento CY8CKIT-059 da Cypress. Essa placa de desenvolvimento carrega um PSOC (*Programable System On Chip*) da família CY8C58LP. O dispositivo presente na placa é o CY8C5888LTI-LP097 que é um dispositivo que carrega um microcontrolador ARM Cortex-M3 em conjunto com periféricos que podem ter suas disposições programadas dentro do componente.

Figura 7 – Planta com Microcontrolador acoplado.



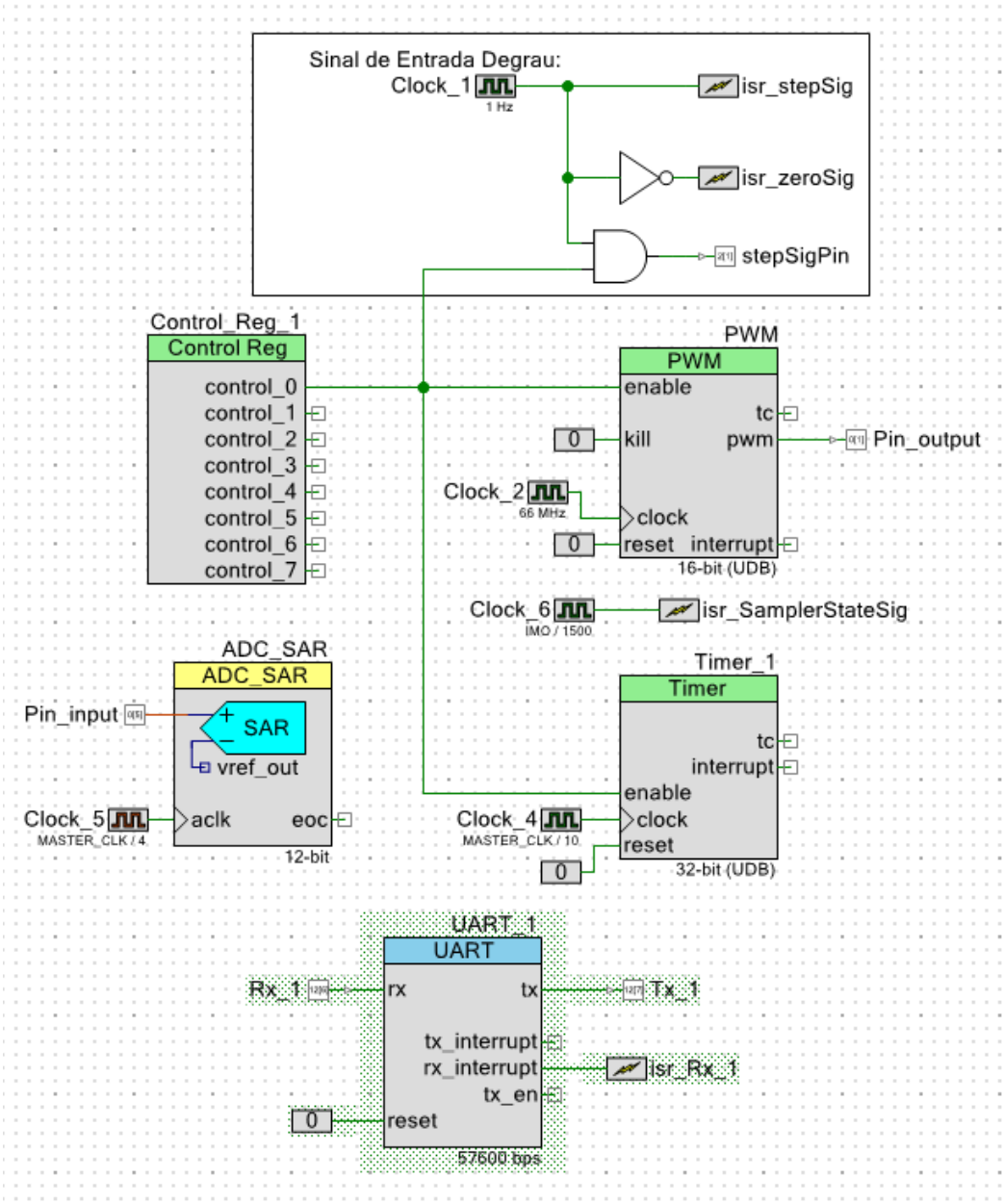
Esse componente possui vários periféricos disponíveis para uso, incluindo periféricos analógicos como amplificadores operacionais, comparadores, PGAs (amplificadores de ganho programável), TIAs (Amplificadores de Trans-impedância), Mixers e outros. Os principais periféricos utilizados nessa atividade foram o ADC SAR de 12 bits e um PWM também de 16 bits configurado exercer um período referente a 12 bits. A Figura 8 retirada do *datasheet* do componente mostra um mapa dos periféricos disponíveis:

Figura 8 – Mapa dos periféricos do PSOC da família CY8C58LP.



A disposição dos periféricos é estabelecida via diagrama de blocos como o esquemático da Figura 9. A partir de um sinal de *clock* de 66 MHz, sinais de *clock* de menor frequência serão gerados para que o PWM seja sincronizado ao sinal de interrupção. Como o período do PWM está configurado para contar até 4096 ciclos, esse exercerá aproximadamente 16113 ações de controle por segundo. Periféricos como a interface UART também foram adicionados como também pode ser visto na Figura 9:

Figura 9 – Esquemático dos periféricos implementados.

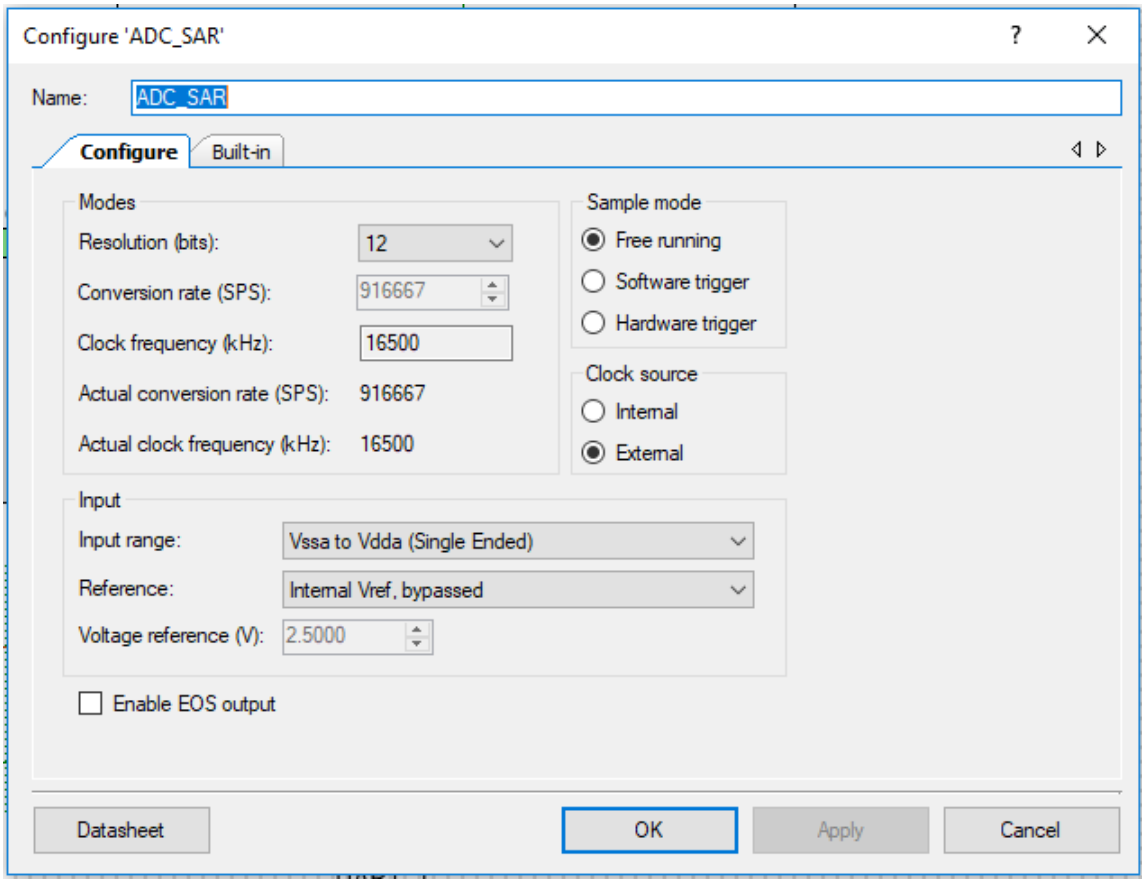



Software: PSoC Creator 4.2

O dispositivo PSOC irá implementar o hardware descrito na Figura 9. O núcleo do processador ARM Cortex-M3 irá interagir com esse hardware por meio dos pinos de interrupção iniciados pela sigla isr. Logo a cada acionamento do pino de interrupção SamplerStateSig, uma função responsável por processar o sinal lido pelo ADC para atualizar o valor da saída PWM será chamada. O sinal lido pelo ADC também é enviado via UART para que possa ser visualizado no computador. O código completo do firmware está disponível no Apêndice C.

O periférico conversor digital do tipo SAR foi configurado como é mostrado na Figura 10. Operando com uma frequência de 16.5 MHz, o periférico está efetuando 916667 leituras por segundo - período de amostragem de 1.09 $\mu$s. Sendo a resolução de 12 bits, o erro de quantização é de $2^{-12} = 0.024\%$ que corresponde a uma relação sinal ruído de 72.25 dB.

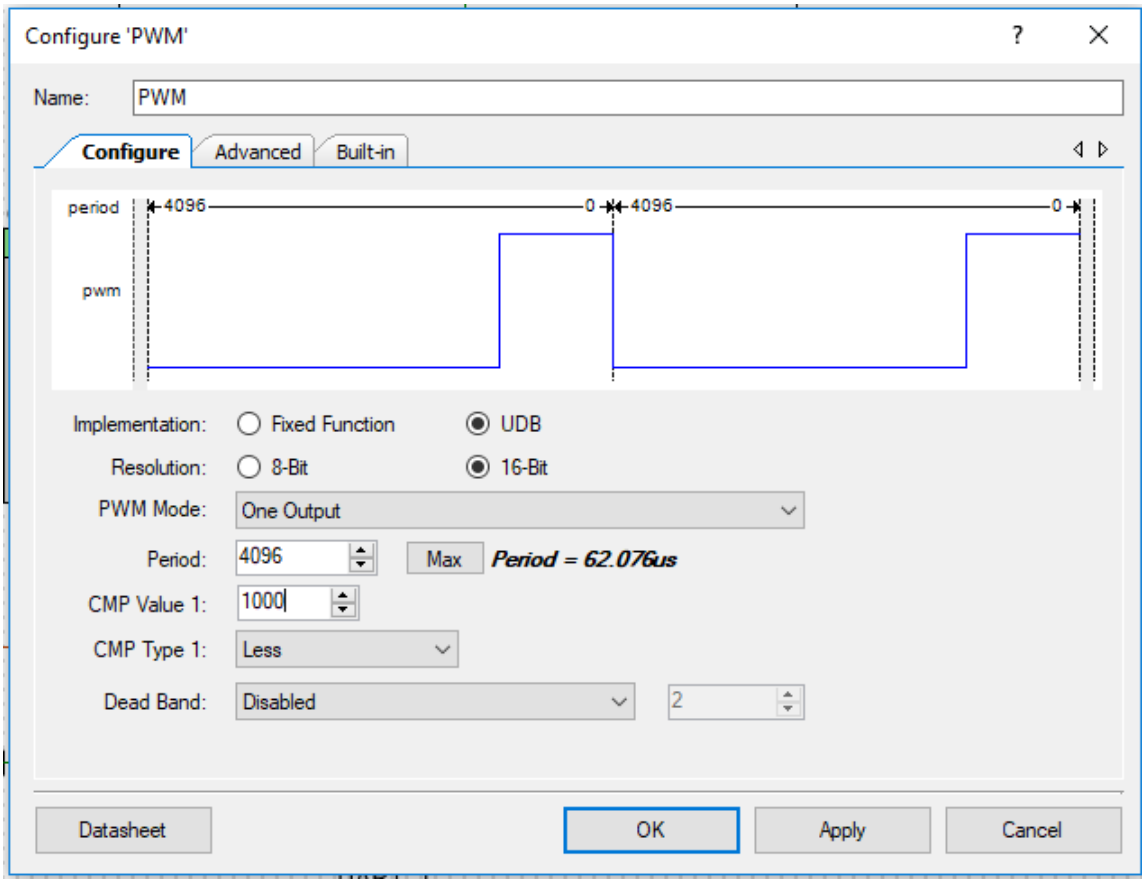
Figura 10 – Medição do sobressinal do sistema de segunda ordem.




Software: PSoC Creator 4.2

Periférico de PWM possui a mesma resolução do ADC utilizado. Funcionando com um *clock* de 66 MHz, período de um ciclo irá corresponder a 62.076 $\mu$s com um *dutycycle* podendo representar 4096 valores diferentes. A configuração do periférico de PWM é apresentada na Figura 11.

Figura 11 – Configuração do PWM.




Software: PSoC Creator 4.2

# 3 Análise da Planta.

## 3.1 Modelagem da Planta.

Para modelar a planta da Figura 12 em uma representação de espaço de estados é preciso definir três variáveis de estado pois o sistema é de terceira ordem (OGATA, 2014). Logo é preciso extrair da planta três equações diferenciais de primeira, uma para cada estado, que englobam esses e suas respectivas derivadas. A equações diferencias extraídas devem se encaixar ao seguinte modelo do espaço de estados:

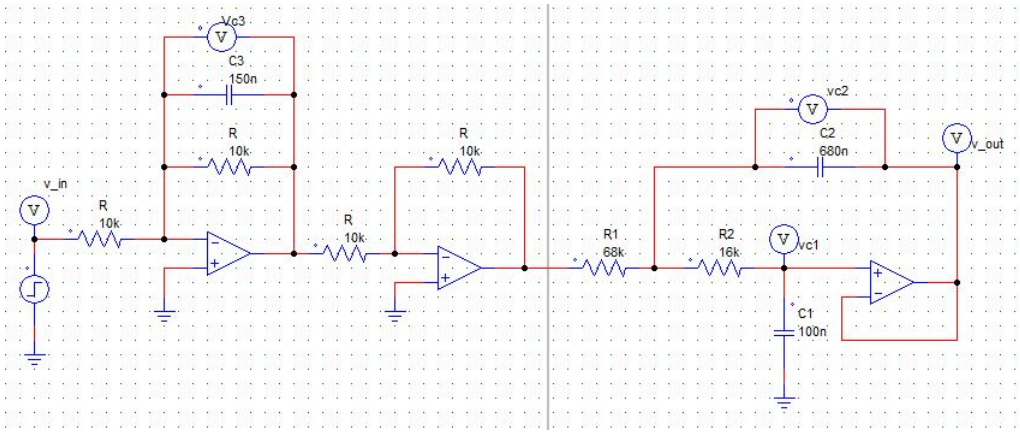
$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = A \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + Bu(t)$$

Se for definido as tensões em cada capacitor como sendo uma variável de estado, a representação no espaço de estados assumirá a seguinte forma:

$$\begin{bmatrix} \dot{V}c_1 \\ \dot{V}c_2 \\ \dot{V}c_3 \end{bmatrix} = A \begin{bmatrix} Vc_1 \\ Vc_2 \\ Vc_3 \end{bmatrix} + Bu(t)$$

Figura 12 – Esquemático da Planta.




Software: PSIM

Com base na Figura 12, por meia das leis das correntes de Kirchhoff, as seguintes equações diferencias de primeira ordem foram extraídas da planta:

$$\dot{V}_{c1} = \frac{V_{c2}}{R_2 C_2}$$

$$\dot{V}_{c2} = -\frac{V_{c1}}{R_1 C_2} - V_{c2}\frac{R_1 + R_2}{R_1 R_2 C_2} + \frac{V_{c3}}{R_1 C_2}$$

$$\dot{V}_{c3} = -\frac{V_{c3}}{RC_3} + \frac{u(t)}{RC_3}$$

$$\begin{bmatrix} \dot{V}c_1 \\ \dot{V}c_2 \\ \dot{V}c_3 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{R_2 C_2} & 0 \\ -\frac{1}{R_1 C_2} & -\frac{R_1+R_2}{R_1 R_2 C_2} & \frac{1}{R_1 C_2} \\ 0 & 0 & -\frac{1}{RC_3} \end{bmatrix} \begin{bmatrix} Vc_1 \\ Vc_2 \\ Vc_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{RC_3} \end{bmatrix} u(t)$$

Além da relação da entrada com os estados do sistema, a representação do espaço de estados requer uma relação dos estados do sistema com a saída:

$$y = \mathbf{C}x + \mathbf{D}u$$

Como pode ser visto na Figura 12, a tensão no capacitor $C_1$ é espelhada para o sinal de saída. Logo os estado $Vc_1$ pode ser definido como saída e assim a relação pode ser definida da seguinte forma:

$$y(t) = x_1 = Vc_1$$

$$y(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} Vc_1 \\ Vc_2 \\ Vc_3 \end{bmatrix}$$

Logo as quatro matrizes da representação no espaço de estados da planta são definidas como:

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{1}{R_2C_2} & 0 \\ -\frac{1}{R_1C_2} & -\frac{R_1+R_2}{R_1R_2C_2} & \frac{1}{R_1C_2} \\ 0 & 0 & -\frac{1}{RC_3} \end{bmatrix} = \begin{bmatrix} 0 & 625 & 0 \\ -21.6263 & -113.5381 & 21.6263 \\ 0 & 0 & -666.6667 \end{bmatrix}$$

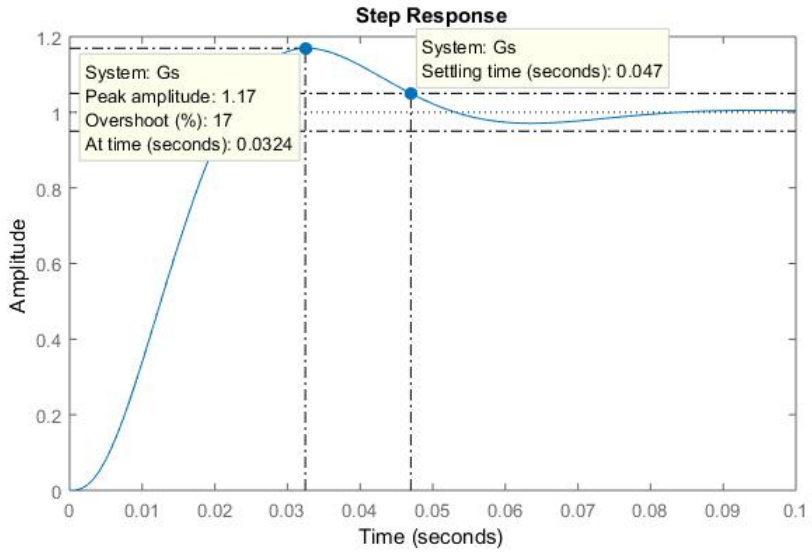
$$\mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{RC_3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 666.6667 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D} = 0$$

Ao analisar esse sistema por meio da função *step()* do MATLAB é possível verificar o comportamento da resposta ao degrau desse sistema:

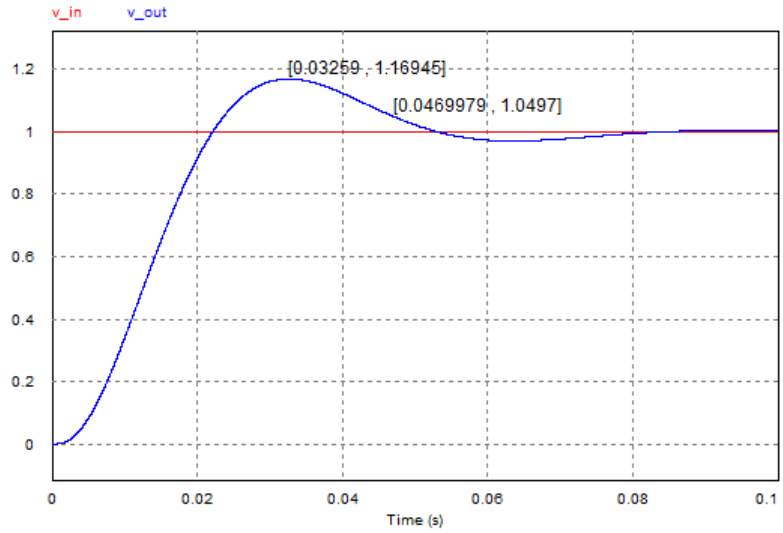
Figura 13 – Resposta ao degrau da representação no espaço de estados.




Software: MATLAB

E a Figura 14, sendo a simulação da planta por meio do *software* PSIM, apresenta um resultado bem semelhante ao da simulação no MATLAB na Figura 13.

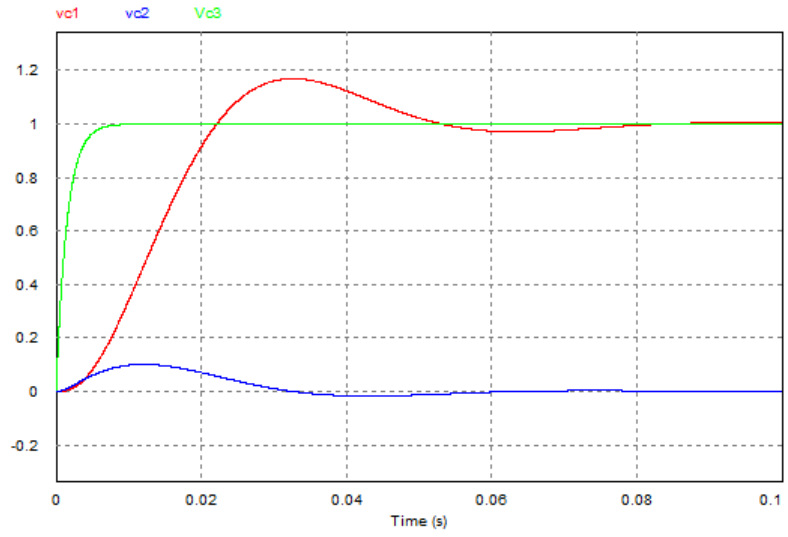
Figura 14 – Resposta ao degrau da planta da Figura 12.




Software: PSIM

Com o PSIM é possível também fazer uma medição dos estados da planta, as tensões em cada capacitor:

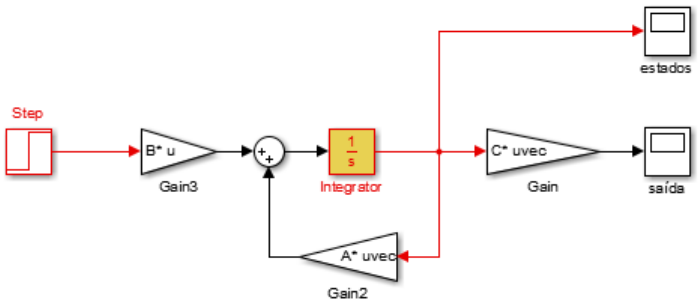
Figura 15 – Resposta ao degrau da planta da Figura 12.




Software: PSIM

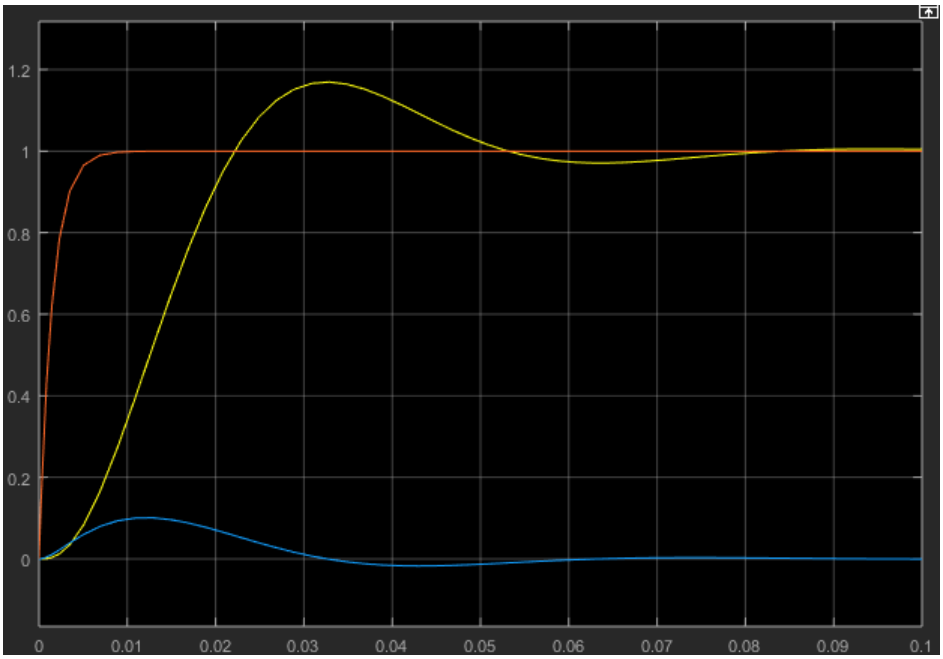
Para comparar os estados da planta simulados no PSIM com os estados do modelo do MATLAB, o simulink permite fazer a leitura dos três estados na saída do integrado do modelo no espaço de estados (Figura 16). É possível ver a semelhança entre a Figura 15 e a Figura 17.

Figura 16 – Modelo no espaço de estados da Figura 12.




Software: MATLAB

Figura 17 – Resposta ao degrau da planta da Figura 12.




Software: MATLAB

## 3.2 Teste de Controlabilidade.

O sistema é dito completamente controlável se o posto da matriz de controlabilidade de dimensão $n$ for igual a $n$(OGATA, 2014):

$$M_{controlabilidade} = \left[ \begin{array}{c:c:c} \mathbf{B} & \mathbf{AB} & \mathbf{A^2B} \end{array} \right]$$

$$\mathbf{AB} = \begin{bmatrix} 0 \\ 1.4428 * 10^4 \\ -4.4444 * 10^5 \end{bmatrix}$$

$$\mathbf{A^2B} = \begin{bmatrix} 9.0109 * 10^6 \\ -1.1125 * 10^7 \\ 2.9629 * 10^8 \end{bmatrix}$$

$$M_{controlabilidade} = \left[ \begin{array}{c:c:c} \mathbf{B} & \mathbf{AB} & \mathbf{A^2B} \end{array} \right] = \left[ \begin{array}{c:c:c} 0 & 0 & 9.0109 * 10^6 \\ 0 & 1.4428 * 10^4 & -1.1125 * 10^7 \\ 666.6667 & -4.4444 * 10^5 & 2.9629 * 10^8 \end{array} \right]$$

Logo como a matriz de controlabilidade possui $n = 3$ e posto 3, o sistema é completamente controlável. Como o sistema possui apenas uma saída, o posto da matriz de controlabilidade de saída precisa ter posto igual a 1. E isso pode ser conferido da seguinte forma:

$$M_{saída} = \left[ \begin{array}{c:c:c} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA^2B} \end{array} \right]$$

$$\mathbf{CB} = \begin{bmatrix} 0 \end{bmatrix}$$

$$\mathbf{CAB} = \begin{bmatrix} 0 \end{bmatrix}$$

$$\mathbf{CA^2B} = \begin{bmatrix} 9.0109 * 10^6 \end{bmatrix}$$

$$M_{saída} = \left[ \begin{array}{c:c:c} \mathbf{CB} & \mathbf{CAB} & \mathbf{CA^2B} \end{array} \right] = \left[ \begin{array}{c:c:c} 0 & 0 & 9.0109 * 10^6 \end{array} \right]$$

## 3.3 Teste de Observabilidade.

O sistema é dito completamente observável se o posto da matriz de observabilidade de dimensão $n$ for igual a $n$ (OGATA, 2014):

$$M_{observabilidade} = \left[ \begin{array}{c:c:c} \mathbf{C}^* & \mathbf{A}^*\mathbf{C}^* & (\mathbf{A}^*)^2\mathbf{C}^* \end{array} \right]$$

$$\mathbf{C}^* = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathbf{A}^*\mathbf{C}^* = \begin{bmatrix} 0 \\ 625 \\ 0 \end{bmatrix}$$

$$(\mathbf{A}^*)^2\mathbf{C}^* = \begin{bmatrix} -1.3516 * 10^4 \\ -7.0961 * 10^4 \\ 1.3516 * 10^4 \end{bmatrix}$$

$$M_{observabilidade} = \left[ \begin{array}{c:c:c} \mathbf{C}^* & \mathbf{A}^*\mathbf{C}^* & (\mathbf{A}^*)^2\mathbf{C}^* \end{array} \right] = \left[ \begin{array}{c:c:c} 1 & 0 & -1.3516 * 10^4 \\ 0 & 625 & -7.0961 * 10^4 \\ 0 & 0 & 1.3516 * 10^4 \end{array} \right]$$

Logo como a matriz de observabilidade possui $n = 3$ e posto 3, dessa forma o sistema é completamente observável. Essa característica indica que será possível implementar um observador de estados (OGATA, 2014).

## 3.4 Erro de Regime Permanente.

Por meio do teorema do valor final é possível verificar o erro de regime permanente por meio da equação de erro da planta no domínio da frequência complexa (B.P.LATHI., 2014):

$$\lim_{s\to 0} sE(s) = e(\infty) = \lim_{s\to 0} sR(s)(1 - \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B})$$

Para verificar o erro de regime permanente para a entrada degrau:$R(s) = \frac{1}{s}$

$$e(\infty) = \lim_{s\to 0}(1 - \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}) = 1 - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$$

$$1 - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} = 1 - \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & -21.6263 & 0 \\ 625 & -113.5381 & 0 \\ 0 & 21.6263 & -666.6667 \end{bmatrix} * \begin{bmatrix} 0 \\ 0 \\ 666.6667 \end{bmatrix} = 2$$

$$e(\infty) = 1 - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} = 2$$

Logo é verificado que a planta possui erro de regime permanente igual a dois. Assim sendo uma planta do tipo 0 (OGATA, 2014).

# 4 Projeto de Controlador no Espaço de Estados por alocação de polos

A ideia básica de um controlador projetado no espaço de estados por realocação de polos consiste por meia da realimentação de estados a alteração da constante de tempo do sistema. É possível visualizar essa ideia entre o paralelo da solução da equação diferencial de primeira ordem na forma escalar e matricial:

$$\dot{x} = ax + bu$$

$$\mathscr{L}\{\dot{x} = bx + au\} = sX(s) - x(0) = aX(s) + bU(s)$$

$$u = -kx$$

$$\dot{x} = ax - bkx = x(a - bk)$$

$$\mathscr{L}\{\dot{x} = x(a - bk)\} = sX(s) - x(0) = (a - bk)X(s)$$

$$X(s) = \frac{x(0)}{(s - a + bk))}$$

$$\mathscr{L}^{-1}\{X(s)\} = x(t) = e^{(a-bk)t}x(0)$$

$$\dot{x} = \mathbf{A}x + \mathbf{B}u$$

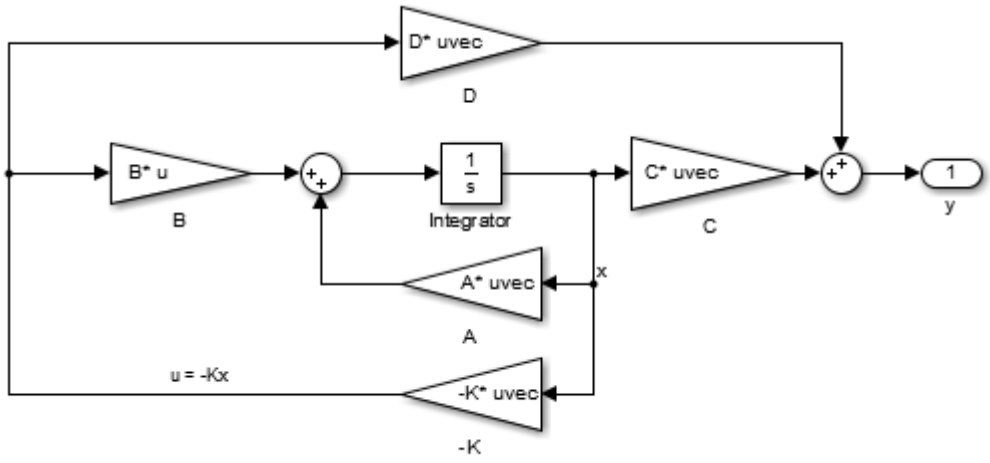
$$u = -\mathbf{K}x$$

$$\dot{x} = \mathbf{A}x - \mathbf{BK}x = (\mathbf{A} - \mathbf{BK})x$$

$$x(t) = e^{(\mathbf{A}-\mathbf{BK})t}x(0)$$

Logo pela realimentação de estados, como apresentado no esquemático da Figura 18, é possível realocar os polos do sistema.

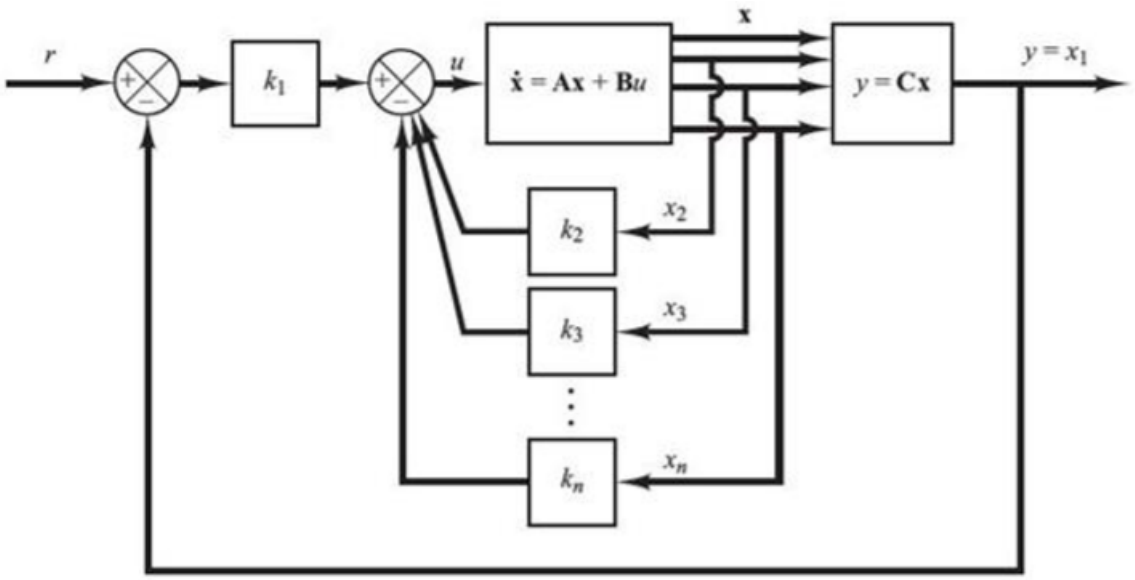
Figura 18 – Esquemático do Sistema Regulador em malha fechada.




Software: Simulink

Para que o sistema possa ter sua referência alterada e deixe de ser um sistema regulador para se tornar um servossistema, o valor de referência $r$ precisa ser modelado junto ao sistema.

Figura 19 – Esquemático do servossistema no espaço de estados.




Livro: (OGATA, 2014)

$$u = -\begin{bmatrix} 0 & k_2 & k_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + k_1(r - x_1) = -\mathbf{K}\mathbf{x} + k_1 r$$

$$\dot{x} = \mathbf{A}x - \mathbf{B}u = x(\mathbf{A} - \mathbf{B}\mathbf{K}) + \mathbf{B}k_1 r$$

$$\dot{x}(t) - \dot{x}(\infty) = (x(t) - x(\infty))(\mathbf{A} - \mathbf{B}\mathbf{K}) + \mathbf{B}k_1(r(t) - r(\infty))$$

$$r(t) = r(\infty)$$

$$\dot{x}(t) - \dot{x}(\infty) = (x(t) - x(\infty))(\mathbf{A} - \mathbf{B}\mathbf{K})$$

$$\dot{e}(t) = (\mathbf{A} - \mathbf{B}\mathbf{K})e(t)$$

Logo se baseando pelo erro, o projeto do servossistema é convertido para um projeto por alocação de polos de um regulador assintoticamente estável (OGATA, 2014).

Como o planta apresenta um erro ao degrau, é necessário a adição de um integrador para que a planta se torne do tipo 1. Isso é possível expandindo a matriz:

$$\begin{bmatrix} \dot{\mathbf{x}}(\mathbf{t}) \\ \dot{\xi}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{-C} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}(\mathbf{t}) \\ \xi(t) \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} r(t)$$

$$\hat{A} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{-C} & 0 \end{bmatrix}$$

$$\hat{B} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{-C} & 0 \end{bmatrix}$$

$$\begin{bmatrix} \dot{x}(t) - \dot{x}(\infty) \\ \dot{\xi}(t) - \dot{\xi}(\infty) \end{bmatrix} = \hat{A} \begin{bmatrix} x(t) - x(\infty) \\ \xi(t) - \xi(\infty) \end{bmatrix} + \hat{B}(u(t) - u(\infty)) + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} (r(t) - r(\infty))$$

$$r(t) = r(\infty)$$

$$\begin{bmatrix} \dot{x}(t) - \dot{x}(\infty) \\ \dot{\xi}(t) - \dot{\xi}(\infty) \end{bmatrix} = \hat{A} \begin{bmatrix} x(t) - x(\infty) \\ \xi(t) - \xi(\infty) \end{bmatrix} + \hat{B}(u(t) - u(\infty))$$

$$\begin{bmatrix} \dot{x}_e(t) \\ \dot{\xi}_e(t) \end{bmatrix} = \hat{A} \begin{bmatrix} x_e(t) \\ \xi_e(t) \end{bmatrix} + \hat{B} u_e(t)$$

$$e(t) = \begin{bmatrix} x_e(t) \\ \xi_e(t) \end{bmatrix}$$

$$\dot{e}(t) = \hat{A}e(t) + \hat{B}u_e(t)$$

$$u_e(t) = -\hat{K}e(t)$$

Assim é possível executar o projeto por alocação de polos:

$$\dot{e}(t) = (\hat{A} - \hat{B}\hat{K})e(t)$$

$$\hat{K} = \left[ \begin{array}{c:c} \mathbf{K} & -k_i \end{array} \right] = \left[ \begin{array}{ccc:c} K_1 & K_2 & K_3 & -k_i \end{array} \right]$$

$$\hat{K} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \left[ \begin{array}{c:c:c:c} \mathbf{B} & \mathbf{AB} & \mathbf{A^2B} & \mathbf{A^3B} \end{array} \right]^{-1} \phi(\mathbf{A})$$

Onde $\phi(\mathbf{A})$ é definido por meio dos polos de determinada localização onde no qual são desejados que sejam dominantes no sistema.

Como é desejador realocar os polos para uma posição desejada, podemos definir essa posição por meio de valores desejados de $\zeta$ e $t_s$. Os polos desejados foram calculados da seguinte forma (OGATA, 2014):

$$\zeta = 0.75$$

$$t_s = 20ms$$

$$\omega_n = \frac{3}{\zeta t_s}$$

$$\mu_{1real} = \mu_{2real} = \zeta\omega_n = 150$$

$$\mu_{1imag} = \omega_n = 214.2857$$

$$\mu_{2imag} = -\omega_n - 214.2857$$

$$j = \begin{bmatrix} \mu_1 & \mu_2 & -1000 & -1000 \end{bmatrix} = \begin{bmatrix} 150 + j214.2857 & 150 - j214.2857 & -1000 & -1000 \end{bmatrix}$$

Como o MATLAB fornece uma função para calcular os ganhos K por meio da fórmula de Arkermann. Logo para os polos desejados:

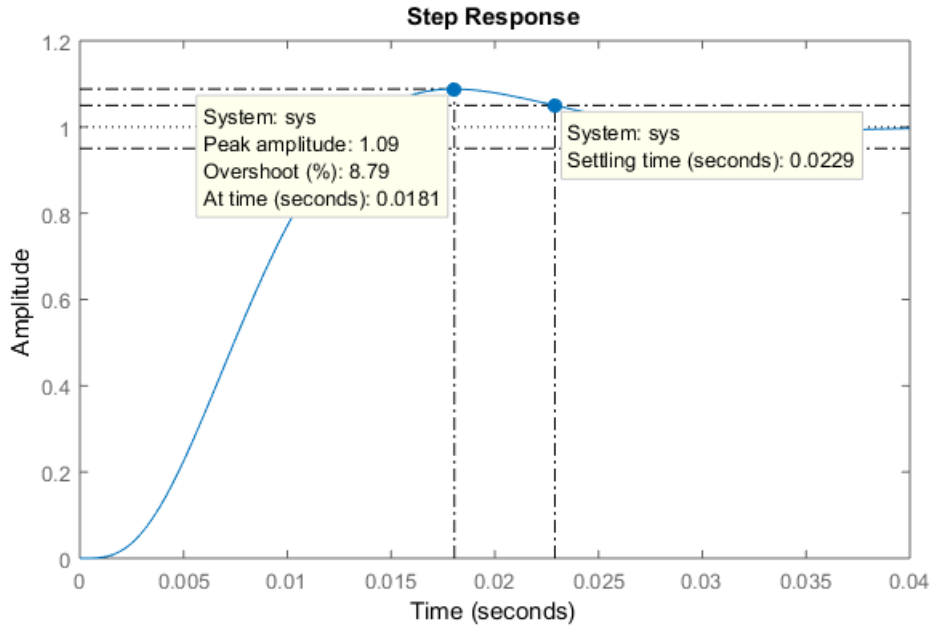
$$j = \begin{bmatrix} \mu_1 & \mu_2 & -1000 & -1000 \end{bmatrix}$$

$$\hat{K} = \left[\begin{array}{cccc:c} 43.8851 & 97.1551 & 2.2797 & 6.9360 * 10^3 \end{array}\right] = \left[\begin{array}{c:c} \mathbf{K} & -k_i \end{array}\right] = \left[\begin{array}{ccc:c} K_1 & K_2 & K_3 & -k_i \end{array}\right]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{A\text{-}BK} & \mathbf{B}k_i \\ \mathbf{\text{-}C} & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} u(t)$$

$$y = \begin{bmatrix} \mathbf{C} & 0 \end{bmatrix} \mathbf{x}$$

Figura 20 – Resposta ao degrau do sistema compensado.




Software: MATLAB

Como no caso dessa planta não é possível fazer a leitura de todos os estados do sistema, é necessário a implementação de um observador de estados. O projeto dessa implementação também pode ser feito por alocação de polos:

$$\dot{\tilde{x}} = \mathbf{A}\tilde{x} + \mathbf{B}u - (y - \mathbf{C}\tilde{x})\mathbf{K}_e = (\mathbf{A} - \mathbf{C}\mathbf{K}_e)\tilde{x} + \mathbf{B}u + \mathbf{K}_e y$$

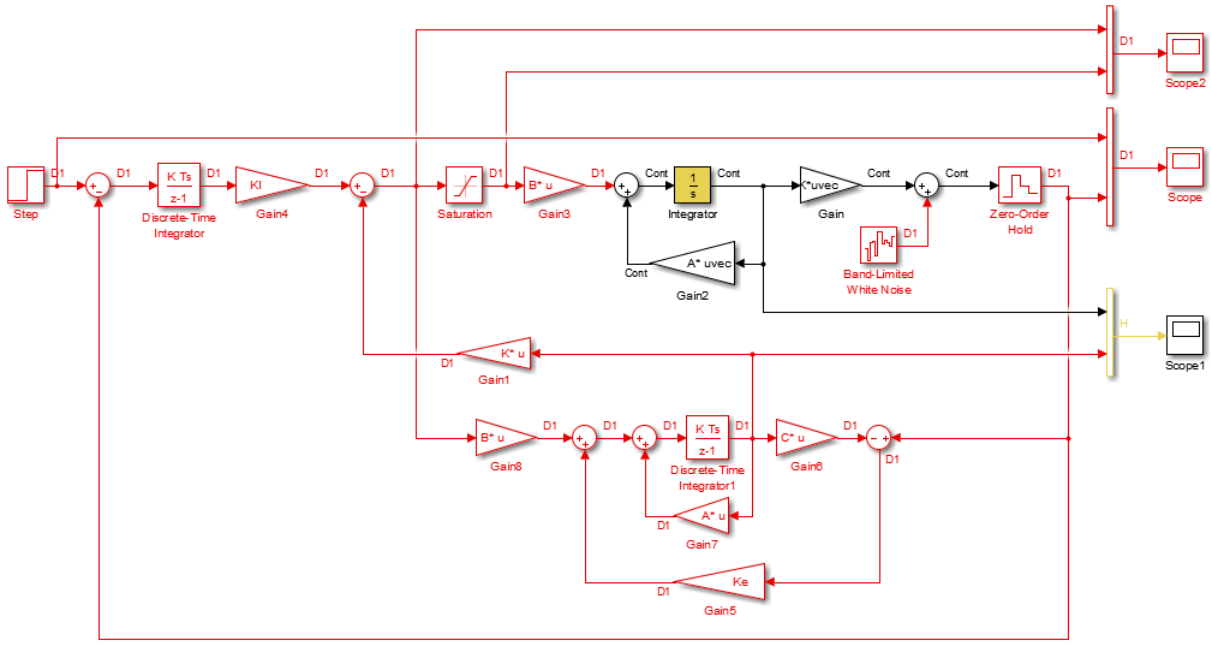
$$\dot{x} - \dot{\tilde{x}} = \mathbf{A}\tilde{x} - \mathbf{A}x - (\mathbf{C}x - \mathbf{C}\tilde{x})\mathbf{K}_e = (\mathbf{A} - \mathbf{C}\mathbf{K}_e)(\tilde{x} - x)$$

$$e = x - \tilde{x}$$

$$\dot{e} = (\mathbf{A} - \mathbf{C}\mathbf{K}_e)e$$

A ideia é que o observador de estados tenha um sinal de erro que o faça seguir os estados da planta. Logo esse sinal de erro deve corrigir os estados observador mais rápido que a dinâmica da planta. Assim seus polos devem ser realocados afim de se tornarem mais rápidos que a planta.

Figura 21 – Resposta ao degrau do sistema compensado.




Software: Simulink

Como o ganho de erro do observador é uma coluna, a matriz A, C e o vetor de polos desejados devem ser utilizadas na forma transposta para que a fórmula de Ackermann retorne um vetor coluna de ganhos. A valor dos polos do observador foram considerados como duas vezes os polos desejados anteriormente calculados.

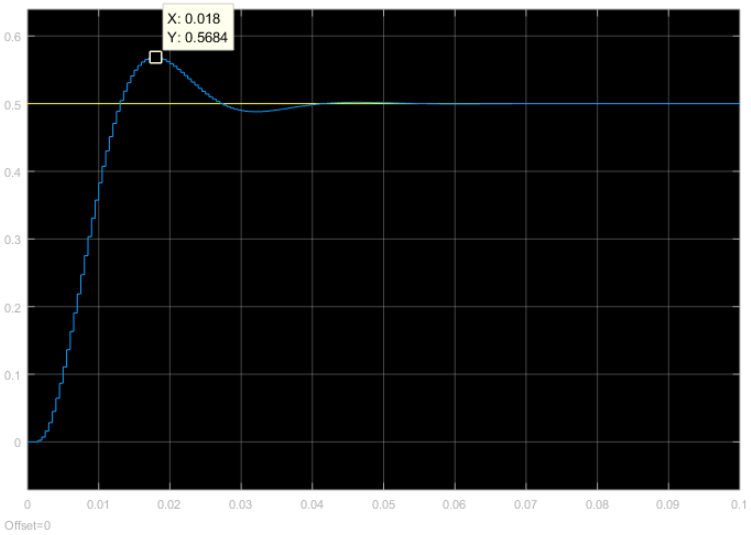
$$j_{obs} = 2 * j'$$

$$\mathbf{K}_e = \begin{bmatrix} 319,7676 \\ 482.0572 \\ 4740.4562 \end{bmatrix}$$

## 4.1 Análise em simulação do controlador.

Observando a simulação da planta com controlador baseado no observador de estados, é possível ver que a amplitude do sobressinal passou um pouco o valor de requisito de projeto.

Figura 22 – Resposta ao degrau do sistema compensado.




Software: Simulink

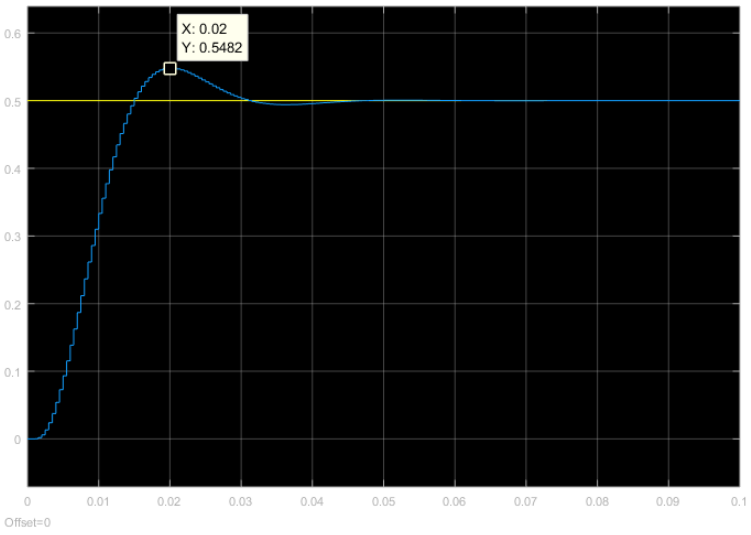
Com o ajuste do valor de $\zeta$ foi possível baixar um pouco o valor do sobressinal para um valor menor que o requisito de projeto:

$$\zeta = 0.85$$

$$t_s = 20ms$$

$$j = \begin{bmatrix} 150 + j176.5 & 150 - j176.5 & -1000 & -1000 \end{bmatrix}$$

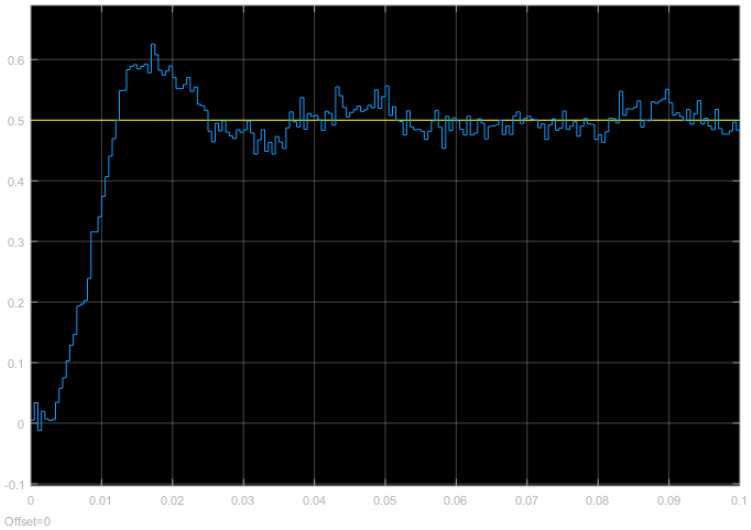
Figura 23 – Resposta ao degrau do sistema compensado após ajuste do $\zeta$.




Software: Simulink

Como pode ser visto na Figura 24, um teste interessante para ser feito no Simulink é do desempenho da controlador com o acréscimo de ruído branco na entrada do ADC.

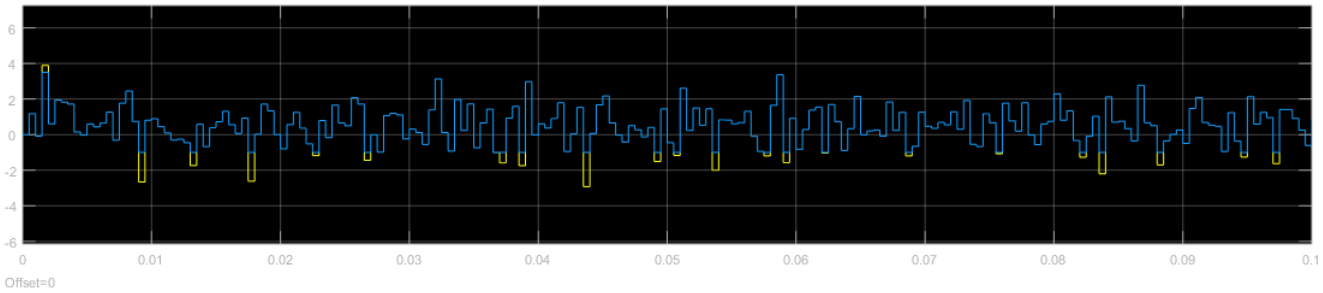
Figura 24 – Resposta ao degrau do sistema compensado com adição de ruído WNG




Software: Simulink

É possível notar que o requisito de projeto deixa de ser atendido com o acréscimo de ruído branco pois a ação de controle começa a atingir amplitudes maiores que a capacidade de representação do PWM. Na Figura 25 é possível ver em amarelo o sinal da ação de controle antes da saturação e em azul depois de ser saturado. Isso fez com que o controlador na atingisse os requisitos de projeto.

Figura 25 – Ação de controle apresentado saturação.




Software: Simulink

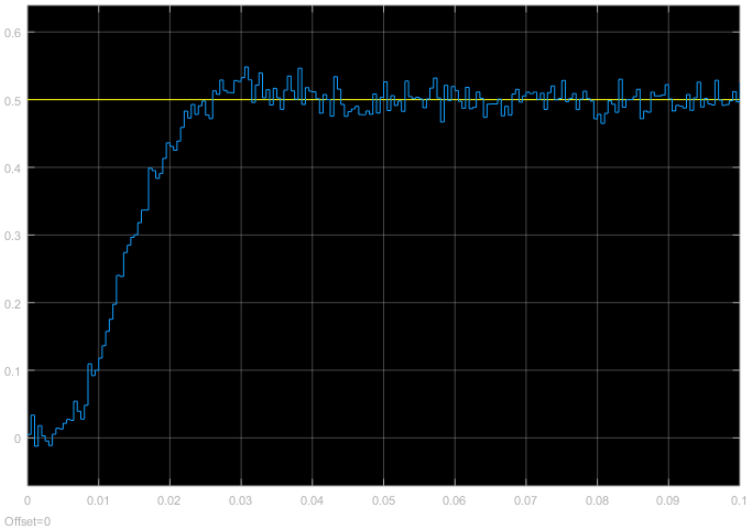
Pelo fato controlador, por ser de quarta ordem, necessitar de quatro polos, mas é desejado apenas um par de polos dominantes, os demais são definidos com valores altos para que não interfiram. Porém se os valores desses polos forem muito grandes a banda passante do sistema se torna proporcionalmente grande fazendo com que o sistema seja mais vulnerável ao ruído. Abaixando o valor desses polos de $-1000$ para $-250$ é possível diminuir a sensibilidade do sistema ao ruído branco. Dessa forma foi resolvido o problema da alta amplitude da ação de controle que ocasionava a saturação do mesmo. Também foi possível baixar o valor de $\zeta$. Isso pode ser verificado na Figura 26 e Figura 27:

$$\zeta = 0.60$$

$$t_s = 20ms$$

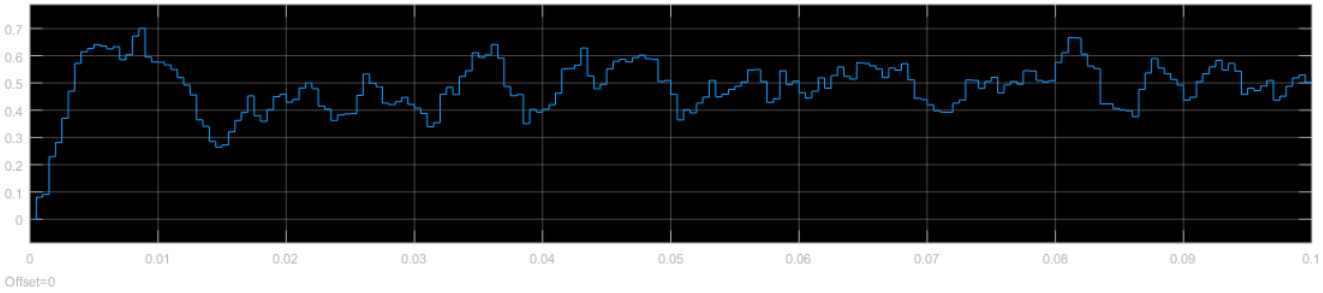
$$j = \begin{bmatrix} 150 + j176.5 & 150 - j176.5 & -250 & -250 \end{bmatrix}$$

Figura 26 – Resposta ao degrau do sistema compensado com polos não dominantes ajustados.




Software: Simulink

Figura 27 – Ação de controle após ajuste de polos não dominantes.




Software: Simulink

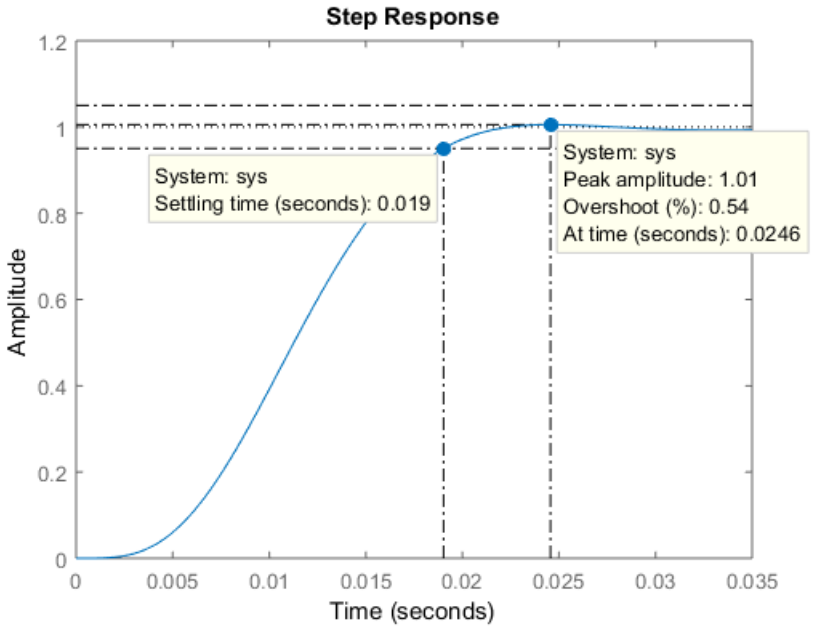
Após ajustes a resposta do sistema apresentou o seguinte sobressinal e tempo de acomodação:

$$\zeta = 0.60$$

$$t_s = 20ms$$

$$j = \begin{bmatrix} 150 + j176.5 & 150 - j176.5 & -250 & -250 \end{bmatrix}$$

Figura 28 – Resposta ao degrau do sistema após ajustes do controlador.



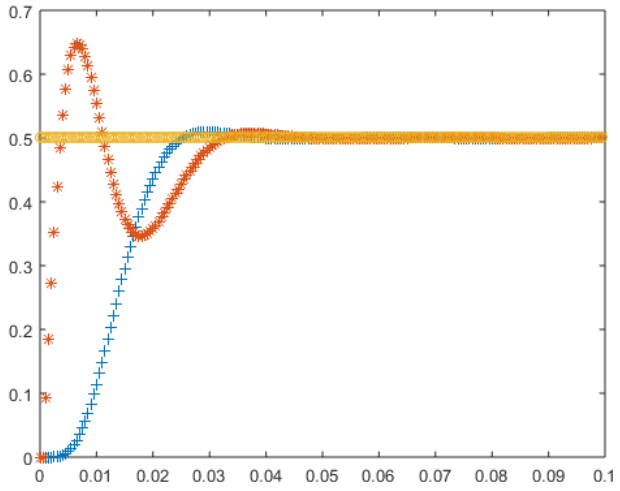

Software: MATLAB

# 5 Implementação recursiva do controlador no Espaço de Estados.

## 5.1 Análise em simulação da equação recursiva.

O Apêndice A apresenta uma implementação de simulação na forma recursiva do controlador para ser executado no MATLAB. Os resultados da simulação: Figura 29 apresenta a resposta ao degrau e ação de controle e a Figura 30 apresenta os estados da planta. Os resultados apresentados são coerentes com as simulações feitas nas seções anteriores.

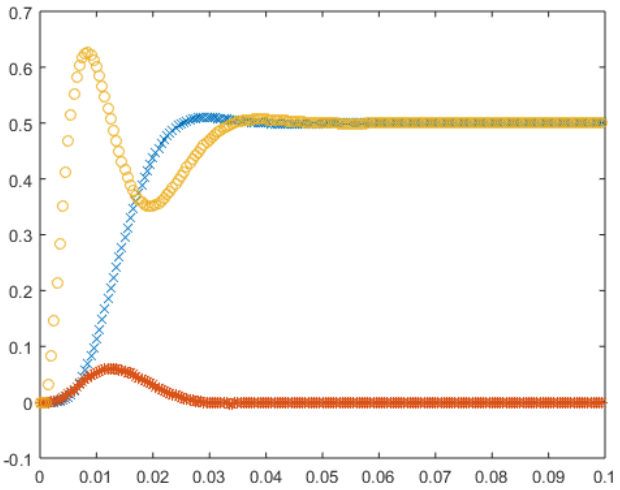
Figura 29 – Resposta ao degrau do sistema compensado.




Software: Simulink

Como o controlador é uma versão discretizada de um projeto de tempo contínuo, a frequência de amostragem de $2kHz$ foi escolhida de tal forma a manter a resolução do estado mais rápido do sistema. Dessa forma o sistema se manteve estável.

Figura 30 – Resposta ao degrau do sistema compensado.




Software: Simulink

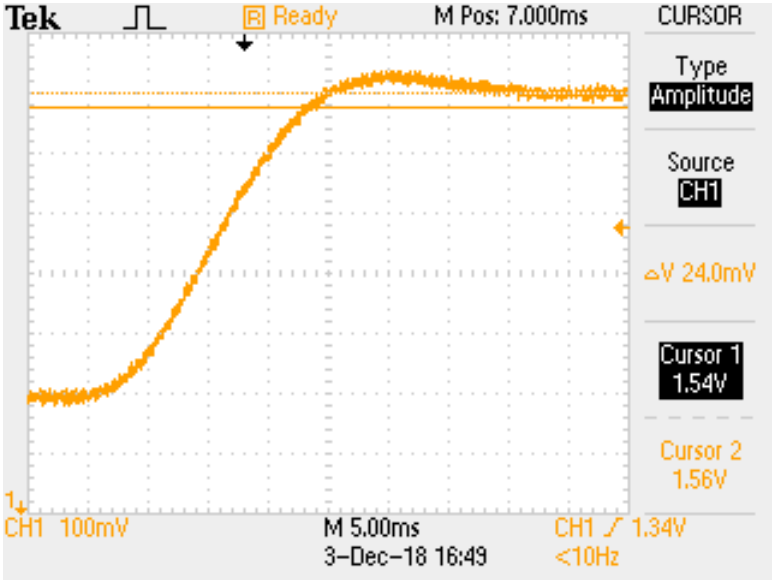
## 5.2 Implementação da equação recursiva com a biblioteca CMSIS.

O pacote CMSIS é um conjunto de bibliotecas desenvolvidas pela Keil otimizadas para os processadores de arquitetura ARM (CMSIS, 2018). Essa biblioteca apresenta um conjunto de ferramentas para operações com matriz que facilitam os projetos de controladores no espaço de estados. O Apêndice B apresenta a implementação do controlador na forma recursiva a ser executado pelo microcontrolador e o Apêndice C apresenta o *firmware* completo.

## 5.3 Análise dos resultados experimentais.

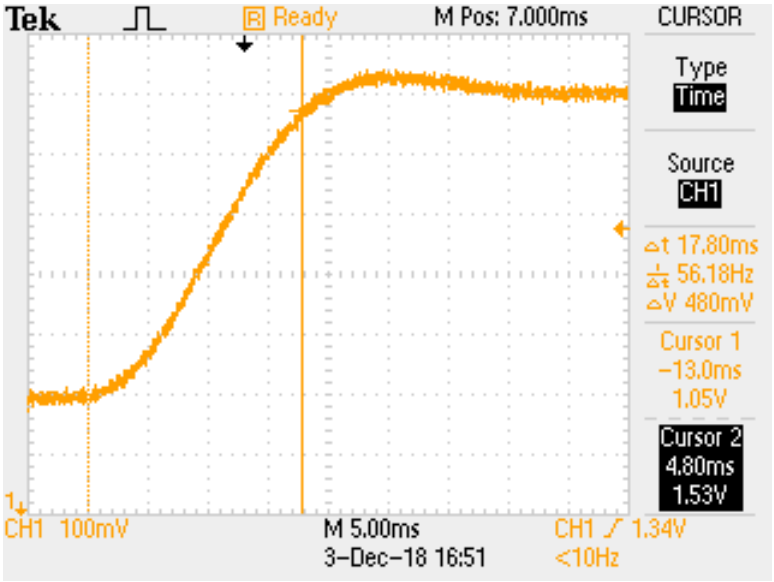
Como o sobressinal apresentado se mostrou com amplitude igual a 5%, foram medidos dois casos: Com tempo de acomodação acontecendo antes do tempo de sobressinal, Figura 31 e Figura 32; e com o tempo de acomodação acontecendo ao mesmo tempo que o sobressinal, Figura 33 e Figura 34.

Figura 31 – Resposta ao degrau do sistema compensado.



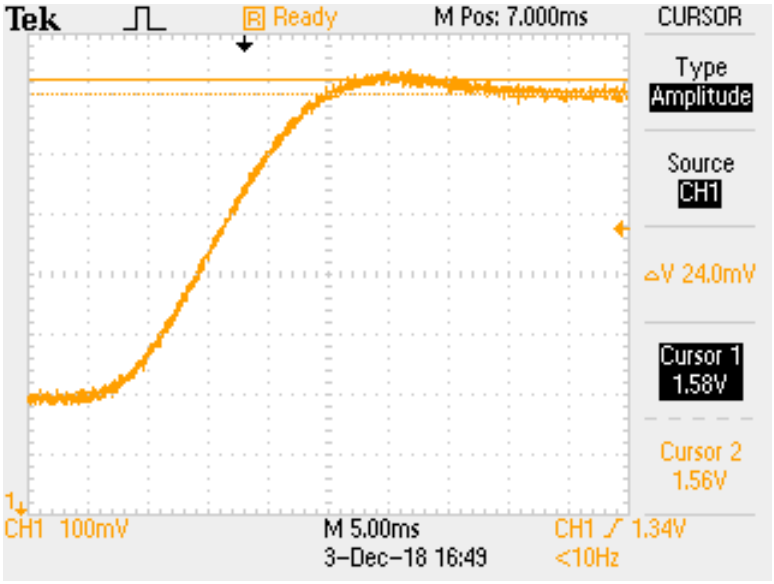

Osciloscópio Tektronics Tds1002c

Figura 32 – Resposta ao degrau do sistema compensado.



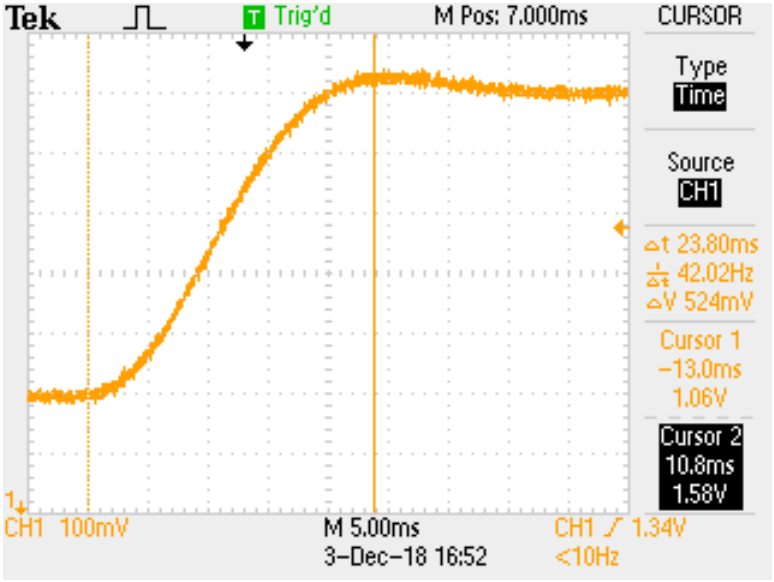

Osciloscópio Tektronics Tds1002c

Figura 33 – Resposta ao degrau do sistema compensado.




Osciloscópio Tektronics Tds1002c

Figura 34 – Resposta ao degrau do sistema compensado.




Osciloscópio Tektronics Tds1002c

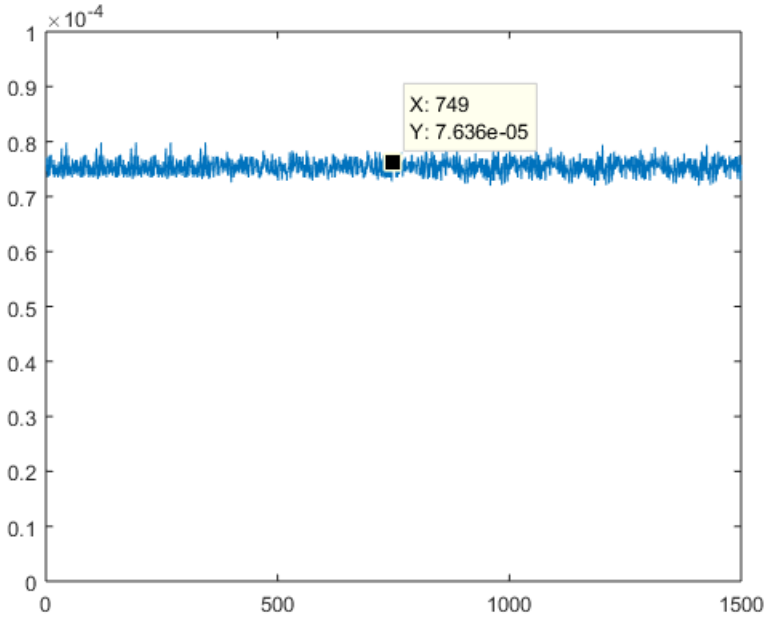
A Tabela 1 apresenta um comparativo entre resultados simulados e experimentais. Para o caso onde a amplitude onde ocorre o sobressinal for considerada como 5%, o projeto extrapolou o requisito de tempo em $30\mu s$. Um pequeno ajuste no sobressinal, para tornar esse um pouco menor que 5%, diminuiria o tempo de acomodação de forma a cumprir o requisito de projeto.

Tabela 1 – Tabela comparativa de resultados.

| | Simulação | Experimental | Experimental | Requisitos | Planta (FTMA) |
|---|---|---|---|---|---|
| $M_p$ (%) | 0.54 | se $M_p < 5\%$ | 5.0 | 10.0 | 20.0 |
| $T_{s5\%}$ (ms) | 19.0 | 17.8 | 23.8 | 23.4 | 46.7 |

A Figura 35 mostra a aquisição do tempo de processamento de cada ciclo de execução do controlador. O tempo de processamento durou aproximadamente $76\mu s$.

Figura 35 – Tempo de processamento.




Software: MATLAB

# 6 Conclusão

Técnicas de projeto de controladores no espaço de estados são uma poderosa ferramento para se trabalhar com sistemas de múltiplas entras e saídas. Porém para sistemas de apenas uma entrada e saída apresentam resultados similares a projetos de controladores discretos com funções transferências no domínio da transformada Z. Se for trabalhada a abordagem da discretização do controlador calculado com variáveis contínuas, há uma necessidade de frequências de amostragens relativamente maiores que os controladores de funções transferências.

Se a planta a ser controlada permitir a leitura de seus estados, a sequência de código a ser processado pelo microcontrolador se torna pequena, porém se surge a necessidade do processamento de um observador de estados, a demanda de processamento aumenta devido ao grande volume de operações com matrizes.

Para aplicações simples controladores baseados em funções transferências parecem se equivaler aos controladores no espaço de estado, porém para aplicações mais complexas o espaço de estado definitivamente será um poderoso caminho.

# Referências

B.P.LATHI. *Sinais e Sistemas Lineares*. [S.l.]: Oxford University Press, Incorporated, 2014. ISBN 9788560031139. Citado na página 15.

CMSIS. *CMSIS-DSP Reference*. [S.l.], 2018. Disponível em: <http://www.keil.com/pack/doc/CMSIS/DSP/html/modules.html>. Citado na página 26.

OGATA, K. *Engenharia de Controle Moderno*. [S.l.]: Pearson Education do Brasil, 2014. ISBN 9788576058106. Citado 6 vezes nas páginas 9, 13, 14, 15, 17 e 19.

# Apêndices

# APÊNDICE A – Controlador recursivo do sistema de controle com observador de ordem plena implementada no MATLAB

```
%% ========================================================================
% Equacao Recursiva: Planta e Observador:
plotSize = 200;
k = 0:plotSize-1; kT = k*T;

r = zeros(1, plotSize); r(1:plotSize) = 0.5;
y = zeros(1, plotSize);
u = zeros(1, plotSize);

erro_I_dot = zeros(1, plotSize);
erro_I = zeros(1, plotSize);

x_obs = zeros(3, plotSize);
x_dot_obs = zeros(3, plotSize);

x_planta = zeros(3, plotSize);
x_dot_planta = zeros(3, plotSize);

for n=2:plotSize
    %%-------------------------------------
    %Integrador para fazer a planta do tipo 0 se tornar do tipo 1:
        erro_I(n) = T*erro_I_dot(n-1)+ erro_I(n-1);
    %%-------------------------------------
    %%-------------------------------------
    %Observador:
        x_obs(:, n) = T*x_dot_obs(:, n-1) + x_obs(:, n-1);
    %%-------------------------------------
    Kx = K*x_obs(:, n);

    %Acao de controle:
    u(n) = KI*erro_I(n) - Kx;

    %%=====================================
    %Planta:
        %%-------------------------------
            x_planta(:, n) = T*x_dot_planta(:, n-1) + x_planta(:, n-1);
        %%-------------------------------
        x_dot_planta(:, n) = B*u(n) + A*x_planta(:, n);
        y(n) = C*x_planta(:, n);%Saida da planta:
    %%=====================================

    erro_I_dot(n) = r(n) - y(n);
    Bu_obs_1 = B*u(n);
    erro_obs = y(n) - C*x_obs(:, n);
    Key = erro_obs*Ke;
    Bu_obs_2 = Bu_obs_1 + Key;
    x_dot_obs(:, n) = Bu_obs_2 + A*x_obs(:, n);
end

figure(6); plot(kT, y, '+'); hold on
plot(kT, u, '*'); hold on
plot(kT, r, 'o')
% ylim([-0.1 1])
hold off

```

# APÊNDICE B – Controlador recursivo do sistema de controle com observador de ordem plena implementada no microcontrolador

```
void sampler_ss_state_process()
{
    //*****************************************************************
    //Counter Start:
    Timer_1_WriteCounter(1000000);
    //*****************************************************************
    main_state = STANDBY_STATE;
    //-----------------------------------------------------------------
    //=================================================================
    erro_I = SAMPLE_RATE*erro_I_dot + erro_I;

    arm_mat_scale_f32(&x_dot_obs, SAMPLE_RATE, &x_dot_obs);
    arm_mat_add_f32(&x_dot_obs, &x_obs, &x_obs);
    arm_mat_mult_f32(&K, &x_obs, &Kx);

    u = KI*erro_I - Kx.pData[0];

    //=================================================================
    outputBuffer = (uint16)(u*0xfff);
    PWM_WriteCompare(outputBuffer);
    adcBuffer = ADC_SAR_GetResult16();
    y=((float32)adcBuffer)/(0xfff);
    //=================================================================

    erro_I_dot = inputBuffer - y;

    arm_mat_scale_f32(&B_obs, u, &Bu_obs_1);

    arm_mat_mult_f32(&C_obs, &x_obs, &y_obs);
    erro_obs = y - y_obs.pData[0];

    arm_mat_scale_f32(&Ke, erro_obs, &Key);
    arm_mat_add_f32(&Bu_obs_1, &Key, &Bu_obs_2);

    arm_mat_mult_f32(&A_obs, &x_obs, &Ax_obs);
    arm_mat_add_f32(&Bu_obs_2, &Ax_obs, &x_dot_obs);
    //-----------------------------------------------------------------
//      uartOut_array[0]=outputBuffer;
//      uartOut_array[1]=outputBuffer>>8;
    uartOut_array[0]=adcBuffer;
    uartOut_array[1]=adcBuffer>>8;
    UART_1_PutArray(uartOut_array, 2);

    //*****************************************************************
    //Counter End:
    cycleCounter = Timer_1_ReadCounter();
    //*****************************************************************

//      uartOut_array[0]=cycleCounter;
//      uartOut_array[1]=cycleCounter>>8;
//      uartOut_array[2]=cycleCounter>>16;
//      uartOut_array[3]=cycleCounter>>24;
//      UART_1_PutArray(uartOut_array, 4);
}

```

# APÊNDICE C – Firmware Completo

```
#include "project.h"
#include "arm_math.h"

// ------------------------------------------------------------------------
#define SAMPLE_RATE 500e-06

#define OFF_COMMAND 0x00
#define FULL_COMMAND  0xff

#define ENABLE_COMMAND 0b00000001
#define RESET_COMMAND  0b00000010

#define CLOSELOOP_COMMAND 0b00000100
#define COMP_ON_COMMAND   0b00001000
#define ZPSYS_ON_COMMAND  0b00010000
uint32 streamSys_status;

uint8 uartOut_array[6];
uint16 adcBuffer, outputBuffer;
uint32 cycleCounter;
float32 adcBufferFloat, erroBuffer, inputBuffer;

// ------------------------------------------------------------------------
// Transfer Function Controller Buffers
float32 Gc_inputBuffer[2], Gc_outputBuffer[2], Gc_K = 2.5389, Gc_beta=-0.3892,
    Gc_alpha=-0.4663;
float32 sys_inputBuffer[3], sys_outputBuffer[3];

// ------------------------------------------------------------------------
// Space State Controller Buffers
float32 r;
float32 erro_I_dot, erro_I, erro_obs;
float32 u;
float32 y;

float32 x_dot_planta_F32[3];
arm_matrix_instance_f32 x_dot_planta;
float32 x_planta_F32[3];
arm_matrix_instance_f32 x_planta;
float32 Bu_planta_F32[3];
arm_matrix_instance_f32 Bu_planta;
float32 Ax_planta_F32[3];
arm_matrix_instance_f32 Ax_planta;
float32 y_planta_F32[1];
arm_matrix_instance_f32 y_planta;

float32 Key_F32[3];
arm_matrix_instance_f32 Key;
const float32 Ke_F32[3]={460.1461, 329.5582, -457.3230};
arm_matrix_instance_f32 Ke;

float32 x_dot_obs_F32[3];
arm_matrix_instance_f32 x_dot_obs;
float32 x_obs_F32[3];
arm_matrix_instance_f32 x_obs;
float32 Bu_obs_1_F32[3];
arm_matrix_instance_f32 Bu_obs_1;
float32 Bu_obs_2_F32[3];
```

```
58 arm_matrix_instance_f32 Bu_obs_2;
59 float32 Ax_obs_F32[3];
60 arm_matrix_instance_f32 Ax_obs;
61 float32 y_obs_F32[1];
62 arm_matrix_instance_f32 y_obs;
63
64
65 const float32 A_F32[9]={0, 625, 0, -21.6263, -113.5381, 21.6263, 0, 0,
       -526.3158};
66 arm_matrix_instance_f32 A_obs;//{0, 1, 0; 0, 0, 1; -7142000, -6.928571000000000e4
       , -6.229300000000000e2};
67 const float32 B_F32[9]={0, 0, 526.3158};
68 arm_matrix_instance_f32 B_obs;
69 const float32 C_F32[9]={1, 0, 0};
70 arm_matrix_instance_f32 C_obs;
71
72 const float32 KI = 601.0964;
73 const float32 K_F32[3]={6.1402, 16.6454, 0.3043};
74 arm_matrix_instance_f32 K;
75 float32 Kx_F32[1];
76 arm_matrix_instance_f32 Kx;
77
78 float32 INT1_BufferX_F32[3], INT1_BufferY_F32[3];
79 arm_matrix_instance_f32 int1_BufferX, int1_BufferY;
80 void arm_mat_integrator_f32(arm_matrix_instance_f32 *pX, arm_matrix_instance_f32
       *pBufferX, arm_matrix_instance_f32 *pY, arm_matrix_instance_f32 *pBufferY);
81
82 float32 INT2_BufferX_F32; float32 INT2_BufferY_F32;
83 void arm_integrator_f32(float32 pX, float32* pBufferX, float32* pY, float32*
       pBufferY);
84
85 void arm_mat_integrator_f32(arm_matrix_instance_f32 *pX, arm_matrix_instance_f32
       *pBufferX, arm_matrix_instance_f32 *pY, arm_matrix_instance_f32 *pBufferY)
86 {
87     arm_mat_scale_f32(pBufferX, SAMPLE_RATE, pBufferX);
88     arm_mat_add_f32(pBufferX, pBufferY, pY);
89     arm_copy_f32(pY->pData, pBufferY->pData, pBufferY->numRows*pBufferY->numCols)
       ;
90     arm_copy_f32(pX->pData, pBufferX->pData, pBufferY->numRows*pBufferY->numCols)
       ;
91 }
92
93 void arm_integrator_f32(float32 pX, float32* pBufferX, float32* pY, float32*
       pBufferY)
94 {
95     *pY = SAMPLE_RATE*(*pBufferX) + *pBufferY;
96     *pBufferY = *pY;
97     *pBufferX = pX;
98 }
99 // ------------------------------------------------------------------------------
100 // State Machine ------------------------------------------------------------------
101 void exit_state_process();
102 void start_state_process();
103 void ss_init_state_process();
104 void standby_state_process();
105 void scanOn_state_process();
106 void scanOff_state_process();
107 void sampler_tf_state_process();
108 void sampler_ss_state_process();
109 void simul_ss_state_process();
110 void simul2_ss_state_process();
111 void closeLoop_state_process();
112 void openLoop_state_process();
```

```
113 void compOn_state_process();
114 void compOff_state_process();
115 void zpsysOn_state_process();
116 void zpsysOff_state_process();
117 void ctrlOn_state_process();
118 void ctrlOff_state_process();
119
120 //Estado comum a todas as m quinas de estado:
121 enum statesOfmachine{
122     EXIT_STATE = 0,
123     START_STATE,
124     SS_INIT_STATE,
125     STANDBY_STATE,
126     SCANON_STATE,
127     SCANOFF_STATE,
128     SAMPLER_TF_STATE,
129     SAMPLER_SS_STATE,
130     SIMUL_SS_STATE,
131     SIMUL2_SS_STATE,
132     CLOSELOOP_STATE,
133     OPENLOOP_STATE,
134     COMPON_STATE,
135     COMPOFF_STATE,
136     ZPSYSON_STATE,
137     ZPSYSOFF_STATE,
138     CTRLON_STATE,
139     CTRLOFF_STATE
140 };
141
142 void (*main_state_table[])()=
143 {
144     exit_state_process,
145     start_state_process,
146     ss_init_state_process,
147     standby_state_process,
148     scanOn_state_process,
149     scanOff_state_process,
150     sampler_tf_state_process,
151     sampler_ss_state_process,
152     simul_ss_state_process,
153     simul2_ss_state_process,
154     closeLoop_state_process,
155     openLoop_state_process,
156     compOn_state_process,
157     compOff_state_process,
158     zpsysOn_state_process,
159     zpsysOff_state_process,
160     ctrlOn_state_process,
161     ctrlOff_state_process
162 };
163
164 volatile int main_state;
165
166 CY_ISR(samplerInterrupt_handler)
167 {
168     main_state =  SAMPLER_SS_STATE;
169 }
170
171 CY_ISR(RxInterrupt_1)
172 {
173     main_state = UART_1_GetChar();
174 }
175
```

```
176 CY_ISR(stepInterrupt_handler)
177 {
178     //inputBuffer = 0.666;
179     inputBuffer = 0.345;
180     //inputBuffer = 1;
181     //inputBuffer = 0.420;
182 }
183
184 CY_ISR(zeroInterrupt_handler)
185 {
186     //inputBuffer = 0.333;
187     inputBuffer = 0.23;
188     //inputBuffer = 0;
189     //inputBuffer = 0.150;
190 }
191
192 int main(void)
193 {
194     CyGlobalIntEnable; /* Enable global interrupts. */
195
196     UART_1_Start();
197     isr_Rx_1_StartEx(RxInterrupt_1);
198
199     isr_SamplerStateSig_StartEx(samplerInterrupt_handler);
200     isr_SamplerStateSig_Disable();
201
202     isr_stepSig_StartEx(stepInterrupt_handler);
203     isr_zeroSig_StartEx(zeroInterrupt_handler);
204
205     Control_Reg_1_Write(OFF_COMMAND);
206
207     while(1)
208     {
209         main_state = START_STATE;
210         while(main_state)
211         {
212             main_state_table[main_state]();
213         }
214     }
215 }
216
217 void exit_state_process()
218 {
219    isr_SamplerStateSig_Disable();
220 }
221
222 void start_state_process()
223 {
224     main_state = SS_INIT_STATE;
225
226     streamSys_status = streamSys_status&(~CLOSELOOP_COMMAND);
227     streamSys_status = streamSys_status&(~COMP_ON_COMMAND);
228     streamSys_status = streamSys_status&(~ZPSYS_ON_COMMAND);
229
230     streamSys_status = streamSys_status|CLOSELOOP_COMMAND;
231     streamSys_status = streamSys_status|COMP_ON_COMMAND;
232     streamSys_status = streamSys_status|ZPSYS_ON_COMMAND;
233
234     Control_Reg_1_Write(OFF_COMMAND);
235
236     Timer_1_Start();
237 }
238
```

```
239 void ss_init_state_process()
240 {
241     main_state = STANDBY_STATE;
242 
243     arm_mat_init_f32(&A_obs, 3, 3, (float32_t *)A_F32);
244     arm_mat_init_f32(&B_obs, 3, 1, (float32_t *)B_F32);
245     arm_mat_init_f32(&C_obs, 1, 3, (float32_t *)C_F32);
246 
247     arm_mat_init_f32(&K, 1, 3, (float32_t *)K_F32);
248     arm_mat_init_f32(&Kx, 1, 1, (float32_t *)Kx_F32);
249     arm_mat_init_f32(&Key, 3, 1, (float32_t *)Key_F32);
250     arm_mat_init_f32(&Ke, 3, 1, (float32_t *)Ke_F32);
251 
252     arm_mat_init_f32(&x_dot_planta, 3, 1, (float32_t *)x_dot_planta_F32);
253     arm_mat_init_f32(&x_planta, 3, 1, (float32_t *)x_planta_F32);
254     arm_mat_init_f32(&Bu_planta, 3, 1, (float32_t *)Bu_planta_F32);
255     arm_mat_init_f32(&Ax_planta, 3, 1, (float32_t *)Ax_planta_F32);
256     arm_mat_init_f32(&y_planta, 1, 1, (float32_t *)y_planta_F32);
257 
258     arm_mat_init_f32(&x_dot_obs, 3, 1, (float32_t *)x_dot_obs_F32);
259     arm_mat_init_f32(&x_obs, 3, 1, (float32_t *)x_obs_F32);
260     arm_mat_init_f32(&Bu_obs_1, 3, 1, (float32_t *)Bu_obs_1_F32);
261     arm_mat_init_f32(&Bu_obs_2, 3, 1, (float32_t *)Bu_obs_2_F32);
262     arm_mat_init_f32(&Ax_obs, 3, 1, (float32_t *)Ax_obs_F32);
263     arm_mat_init_f32(&y_obs, 1, 1, (float32_t *)y_obs_F32);
264 
265     arm_mat_init_f32(&int1_BufferX, 3, 1, (float32_t *)INT1_BufferX_F32);
266     arm_mat_init_f32(&int1_BufferY, 3, 1, (float32_t *)INT1_BufferY_F32);
267 }
268 
269 void standby_state_process()
270 {
271 
272 }
273 
274 void scanOn_state_process()
275 {
276     main_state = STANDBY_STATE;
277 
278     isr_SamplerStateSig_Enable();
279 
280     isr_stepSig_Enable();
281     isr_zeroSig_Enable();
282 
283     ADC_SAR_Start();
284     ADC_SAR_StartConvert();
285     PWM_Start();
286     PWM_WriteCompare(10);
287 
288     Control_Reg_1_Write(ENABLE_COMMAND);
289 }
290 
291 void scanOff_state_process()
292 {
293     main_state = STANDBY_STATE;
294 
295     isr_SamplerStateSig_Disable();
296 
297     ADC_SAR_Stop();
298     ADC_SAR_StopConvert();
299 
300     PWM_Stop();
301 
```

```
302     Control_Reg_1_Write(OFF_COMMAND);
303 }
304
305 #define CLOSELOOP_COMMAND 0b00000100
306 #define COMP_ON_COMMAND   0b00001000
307 #define ZPSYS_ON_COMMAND 0b00010000
308
309 void sampler_tf_state_process()
310 {
311     //*****************************************************************
312     //Counter Start:
313     Timer_1_WriteCounter(1000000);
314     //*****************************************************************
315
316     main_state = STANDBY_STATE;
317
318     //-----------------------------------------------------------------
319     adcBuffer = ADC_SAR_GetResult16();
320     adcBufferFloat=((float32)adcBuffer)/(0xfff);
321
322     //-----------------------------------------------------------------
323     //Opened-Loop or Closed-Loop:
324     if(streamSys_status&CLOSELOOP_COMMAND)
325         erroBuffer = inputBuffer - adcBufferFloat;//Closed-Loop
326     else
327         erroBuffer = inputBuffer;
328     //-----------------------------------------------------------------
329
330     Gc_inputBuffer[0] = erroBuffer;
331
332     //-----------------------------------------------------------------
333     //Compensator ON or OFF:
334     if(streamSys_status&COMP_ON_COMMAND)
335         Gc_outputBuffer[0] = Gc_K*Gc_inputBuffer[0] + Gc_K*Gc_alpha*
        Gc_inputBuffer[1] - Gc_beta*Gc_outputBuffer[1];//ON
336     else
337         Gc_outputBuffer[0] = Gc_inputBuffer[0];//OFF
338     //-----------------------------------------------------------------
339     sys_inputBuffer[0] = Gc_outputBuffer[0];
340     //-----------------------------------------------------------------
341     if(streamSys_status&ZPSYS_ON_COMMAND)
342         sys_outputBuffer[0] =  sys_inputBuffer[0] - 1.478*sys_inputBuffer[1] +
        0.6531*sys_inputBuffer[2] + sys_outputBuffer[1];//ON
343     else
344         sys_outputBuffer[0] = sys_inputBuffer[0];//OFF
345     //-----------------------------------------------------------------
346
347     outputBuffer = (uint16)(sys_outputBuffer[0]*0xfff);
348     PWM_WriteCompare(outputBuffer);
349
350
351
352     Gc_inputBuffer[1] = Gc_inputBuffer[0];
353     Gc_outputBuffer[1] = Gc_outputBuffer[0];
354
355     sys_inputBuffer[2] = sys_inputBuffer[1];
356     sys_outputBuffer[2] = sys_outputBuffer[1];
357     sys_inputBuffer[1] = sys_inputBuffer[0];
358     sys_outputBuffer[1] = sys_outputBuffer[0];
359     //-----------------------------------------------------------------
360
361
362
```

```
363 //      uartOut_array[0]=outputBuffer;
364 //      uartOut_array[1]=outputBuffer>>8;
365
366     uartOut_array[0]=adcBuffer;
367     uartOut_array[1]=adcBuffer>>8;
368
369     //*****************************************************************
370     //Counter End:
371     cycleCounter = Timer_1_ReadCounter();
372     //*****************************************************************
373
374 //      uartOut_array[2]=cycleCounter;
375 //      uartOut_array[3]=cycleCounter>>8;
376 //      uartOut_array[4]=cycleCounter>>16;
377 //      uartOut_array[5]=cycleCounter>>24;
378 //
379     UART_1_PutArray(uartOut_array, 2);
380 }
381
382 void sampler_ss_state_process()
383 {
384     //*****************************************************************
385     //Counter Start:
386     Timer_1_WriteCounter(1000000);
387     //*****************************************************************
388
389     main_state = STANDBY_STATE;
390     //-----------------------------------------------------------------
391     //=================================================================
392     erro_I = SAMPLE_RATE*erro_I_dot + erro_I;
393
394     arm_mat_scale_f32(&x_dot_obs, SAMPLE_RATE, &x_dot_obs);
395     arm_mat_add_f32(&x_dot_obs, &x_obs, &x_obs);
396     arm_mat_mult_f32(&K, &x_obs, &Kx);
397
398     u = KI*erro_I - Kx.pData[0];
399
400 //      u = inputBuffer;
401
402     //=================================================================
403     outputBuffer = (uint16)(u*0xfff);
404     PWM_WriteCompare(outputBuffer);
405     adcBuffer = ADC_SAR_GetResult16();
406     y=((float32)adcBuffer)/(0xfff);
407     //=================================================================
408
409     erro_I_dot = inputBuffer - y;
410
411     arm_mat_scale_f32(&B_obs, u, &Bu_obs_1);
412
413     arm_mat_mult_f32(&C_obs, &x_obs, &y_obs);
414     erro_obs = y - y_obs.pData[0];
415
416     arm_mat_scale_f32(&Ke, erro_obs, &Key);
417     arm_mat_add_f32(&Bu_obs_1, &Key, &Bu_obs_2);
418
419     arm_mat_mult_f32(&A_obs, &x_obs, &Ax_obs);
420     arm_mat_add_f32(&Bu_obs_2, &Ax_obs, &x_dot_obs);
421     //-----------------------------------------------------------------
422
423
424
425 //      uartOut_array[0]=outputBuffer;
```

```
426 //      uartOut_array[1]=outputBuffer>>8;
427
428     uartOut_array[0]=adcBuffer;
429     uartOut_array[1]=adcBuffer>>8;
430
431     UART_1_PutArray(uartOut_array, 2);
432
433     //*****************************************************************
434     //Counter End:
435     cycleCounter = Timer_1_ReadCounter();
436     //*****************************************************************
437
438 //      uartOut_array[0]=cycleCounter;
439 //      uartOut_array[1]=cycleCounter>>8;
440 //      uartOut_array[2]=cycleCounter>>16;
441 //      uartOut_array[3]=cycleCounter>>24;
442 ////
443 //      UART_1_PutArray(uartOut_array, 4);
444 }
445
446 void simul_ss_state_process()
447 {
448     //*****************************************************************
449     //Counter Start:
450     Timer_1_WriteCounter(1000000);
451     //*****************************************************************
452
453     main_state = STANDBY_STATE;
454     //-----------------------------------------------------------------
455     //=================================================================
456     //&x_dot_obs int -> &x_obs
457 //      arm_mat_integrator_f32(&x_dot_obs, &int1_BufferX, &x_obs, &int1_BufferY);
458     arm_mat_scale_f32(&x_dot_planta, SAMPLE_RATE, &x_dot_planta);
459     arm_mat_add_f32(&x_dot_planta, &x_planta, &x_planta);
460
461     arm_mat_scale_f32(&B_obs, inputBuffer, &Bu_planta);
462
463     arm_mat_mult_f32(&A_obs, &x_planta, &Ax_planta);
464
465     arm_mat_add_f32(&Bu_planta, &Ax_planta, &x_dot_planta);
466
467     arm_mat_mult_f32(&C_obs, &x_planta, &y_planta);
468
469     //=================================================================
470
471     adcBuffer=(uint16)(y_planta.pData[0]*0xfff);
472     //-----------------------------------------------------------------
473
474 //      uartOut_array[0]=outputBuffer;
475 //      uartOut_array[1]=outputBuffer>>8;
476
477     uartOut_array[0]=adcBuffer;
478     uartOut_array[1]=adcBuffer>>8;
479
480     UART_1_PutArray(uartOut_array, 2);
481
482     //*****************************************************************
483     //Counter End:
484     cycleCounter = Timer_1_ReadCounter();
485     //*****************************************************************
486
487     uartOut_array[0]=cycleCounter;
488     uartOut_array[1]=cycleCounter>>8;
```

```
489     uartOut_array[2]=cycleCounter>>16;
490     uartOut_array[3]=cycleCounter>>24;
491 
492     UART_1_PutArray(uartOut_array, 4);
493 }
494 
495 void simul2_ss_state_process()
496 {
497     //*****************************************************************
498     //Counter Start:
499     Timer_1_WriteCounter(1000000);
500     //*****************************************************************
501 
502     main_state = STANDBY_STATE;
503     //-----------------------------------------------------------------
504     //=================================================================
505     erro_I = SAMPLE_RATE*erro_I_dot + erro_I;
506 
507     arm_mat_scale_f32(&x_dot_obs, SAMPLE_RATE, &x_dot_obs);
508     arm_mat_add_f32(&x_dot_obs, &x_obs, &x_obs);
509     arm_mat_mult_f32(&K, &x_obs, &Kx);
510 
511     u = KI*erro_I - Kx.pData[0];
512     outputBuffer=(uint16)(u*0xfff);
513     //=================================================================
514     //&x_dot_obs int -> &x_obs
515     arm_mat_scale_f32(&x_dot_planta, SAMPLE_RATE, &x_dot_planta);
516     arm_mat_add_f32(&x_dot_planta, &x_planta, &x_planta);
517 
518     arm_mat_scale_f32(&B_obs, u, &Bu_planta);
519     arm_mat_mult_f32(&A_obs, &x_planta, &Ax_planta);
520     arm_mat_add_f32(&Bu_planta, &Ax_planta, &x_dot_planta);
521 
522     arm_mat_mult_f32(&C_obs, &x_planta, &y_planta);
523     //=================================================================
524 
525     erro_I_dot = inputBuffer - y_planta.pData[0];
526 
527     arm_mat_scale_f32(&B_obs, u, &Bu_obs_1);
528 
529     arm_mat_mult_f32(&C_obs, &x_obs, &y_obs);
530     erro_obs = y_planta.pData[0] - y_obs.pData[0];
531 
532     arm_mat_scale_f32(&Ke, erro_obs, &Key);
533     arm_mat_add_f32(&Bu_obs_1, &Key, &Bu_obs_2);
534 
535     arm_mat_mult_f32(&A_obs, &x_obs, &Ax_obs);
536     arm_mat_add_f32(&Bu_obs_2, &Ax_obs, &x_dot_obs);
537     //=================================================================
538 
539     adcBuffer=(uint16)(y_planta.pData[0]*0xfff);
540     //-----------------------------------------------------------------
541 
542     uartOut_array[0]=outputBuffer;
543     uartOut_array[1]=outputBuffer>>8;
544 
545 //     uartOut_array[0]=adcBuffer;
546 //     uartOut_array[1]=adcBuffer>>8;
547 
548     UART_1_PutArray(uartOut_array, 2);
549 
550     //*****************************************************************
551     //Counter End:
```

```
552     cycleCounter = Timer_1_ReadCounter();
553     //************************************************************
554 
555 //      uartOut_array[0]=cycleCounter;
556 //      uartOut_array[1]=cycleCounter>>8;
557 //      uartOut_array[2]=cycleCounter>>16;
558 //      uartOut_array[3]=cycleCounter>>24;
559 ////
560 //      UART_1_PutArray(uartOut_array, 4);
561 }
562 
563 void closeLoop_state_process()
564 {
565     main_state = STANDBY_STATE;
566     streamSys_status = streamSys_status|CLOSELOOP_COMMAND;
567 }
568 
569 void openLoop_state_process()
570 {
571     main_state = STANDBY_STATE;
572     streamSys_status = streamSys_status&(~CLOSELOOP_COMMAND);
573 }
574 
575 void compOn_state_process()
576 {
577     main_state = STANDBY_STATE;
578     streamSys_status = streamSys_status|COMP_ON_COMMAND;
579 }
580 
581 void compOff_state_process()
582 {
583     main_state = STANDBY_STATE;
584     streamSys_status = streamSys_status&(~COMP_ON_COMMAND);
585 }
586 
587 void zpsysOn_state_process()
588 {
589     main_state = STANDBY_STATE;
590     streamSys_status = streamSys_status|ZPSYS_ON_COMMAND;
591 }
592 
593 void zpsysOff_state_process()
594 {
595     main_state = STANDBY_STATE;
596     streamSys_status = streamSys_status&(~ZPSYS_ON_COMMAND);
597 }
598 
599 void ctrlOn_state_process()
600 {
601     main_state = STANDBY_STATE;
602     streamSys_status = streamSys_status|CLOSELOOP_COMMAND;
603     streamSys_status = streamSys_status|COMP_ON_COMMAND;
604     streamSys_status = streamSys_status|ZPSYS_ON_COMMAND;
605 }
606 
607 void ctrlOff_state_process()
608 {
609     main_state = STANDBY_STATE;
610     streamSys_status = streamSys_status&(~CLOSELOOP_COMMAND);
611     streamSys_status = streamSys_status&(~COMP_ON_COMMAND);
612     streamSys_status = streamSys_status&(~ZPSYS_ON_COMMAND);
613 }
614 
```